OSDT China
Beijing 2019

In-Kernel VM & Service

Feng Li (李枫)
hkli2013@126.com
Nov 9，2019

Agenda

I. eBPF
- What is it
- Development
- Application

II. Testbed
- Development Board
- Software Platform

III. eBPF-based In-Kernel Service
- Service function chaining
- Polycube

IV. Lua-based In-Kernel VM
- NetBSD
- Linux

V. Other In-Kernel VMs
- KPlugs
- WASM

VI. New Ideas

- LuaJIT
- NtopNG
- P4 & Stratum
- LISA2

VII. Wrap-up

I. eBPF

1) What is it

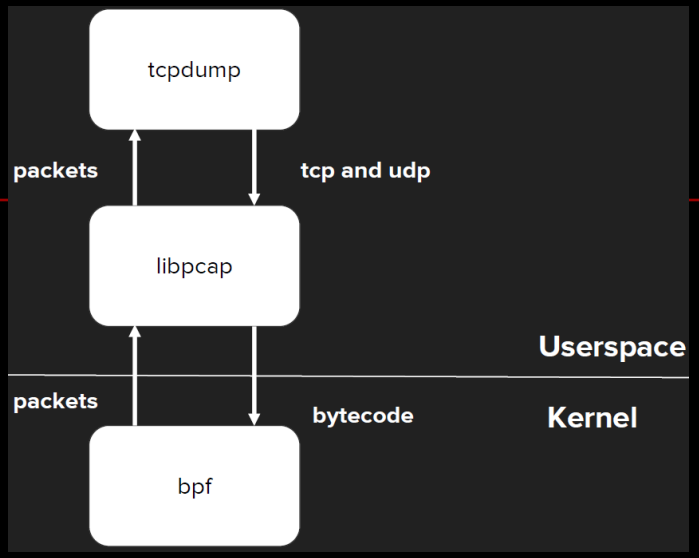
1.1 BPF (Berkeley Packet Filter, aka cBPF)

- https://en.wikipedia.org/wiki/Berkeley_Packet_Filter
- http://www.tcpdump.org/papers/bpf-usenix93.pdf
- History
  - Before BPF, each OS (Sun, DEC, SGI etc) had its own packet filtering API
  - In 1993: Steven McCanne & Van Jacobsen released a paper titled the *BSD Packet Filter (BPF)*
  - Implemented as “Linux Socket Filter” in kernel 2.2
  - While maintaining the BPF language (for describing filters), uses a different internal architecture

Source: ebpfbasics-190611051559.pdf

What is it

- Network packet filtering, Seccomp
- Filter Expressions → Bytecode → Interpret
- Small, in-kernel VM, Register based, switch dispatch interpreter, few instructions
- BPF uses a simple, non-shared buffer model made possible by today's larger address space



Source: ebpfbasics-190611051559.pdf

- Bytecode, register based VM, with a limited instruction set
- Runs in-kernel, designed for fast packet filtering
- 32-bit instructions (LOAD, STORE, ALU, BRANCH, RETURN)
- 2, 32-bit registers (A, X), hidden frame pointer

Source: understandingebpfinahurry-190611040804.pdf

https://blog.cloudflare.com/bpf-the-forgotten-bytecode/

…

1.2 eBPF (extended BPF)

- since Linux Kernel v3.15 and ongoing
- aims at being a universal in-kernel virtual machine
- a simple way to extend the functionality of Kernel at runtime
- "Dtrace( ) for Linux"

BPF Features by Linux Kernel Version

- https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md

Instruction Set

- https://github.com/iovisor/bpf-docs/blob/master/eBPF.md

Projects using eBPF

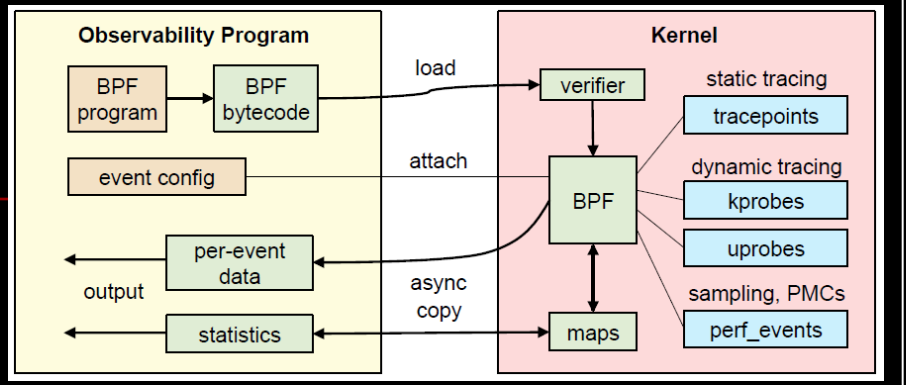
- http://cilium.readthedocs.io/en/latest/bpf/#projects-using-bpf

Good Resource

- https://github.com/zoidbergwill/awesome-ebpf

What is it

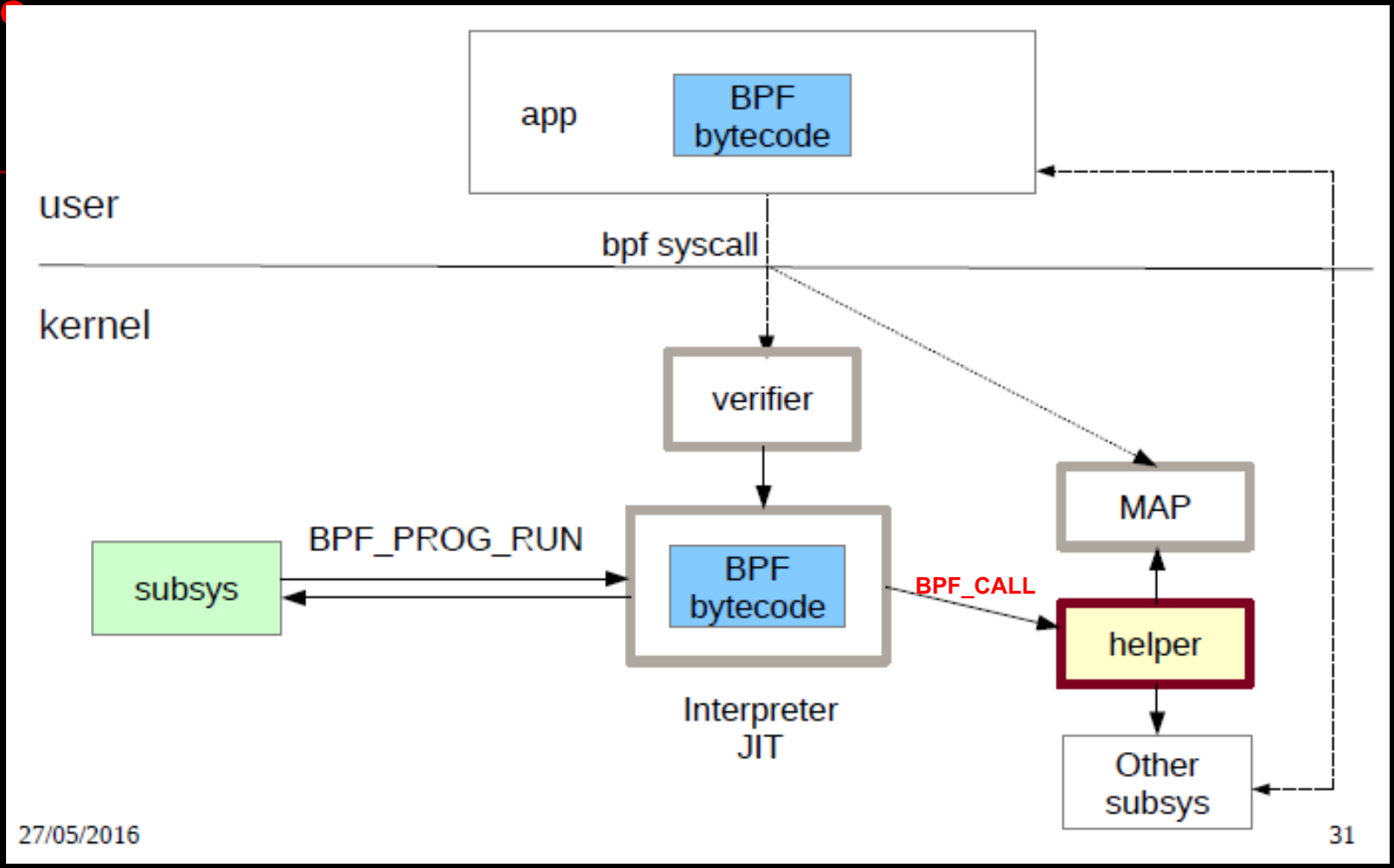
- User-defined, sandboxed bytecode executed by the kernel
- VM that implements a RISC-like assembly language in kernel space



- Similar to LSF, but with the following improvements:
  - More registers, JIT compiler (flexible/ faster), verifier
  - Attach on Tracepoint, Kprobe, Uprobe, USDT
  - In-kernel trace aggregation & filtering
  - Control via bpf()
  - Designed for general event processing within the kernel
  - All interactions between kernel/ user space are done through eBPF “maps”

Source: ebpfbasics-190611051559.pdf
Source: https://kernel-recipes.org/en/2017/talks/performance-analysis-with-bpf/

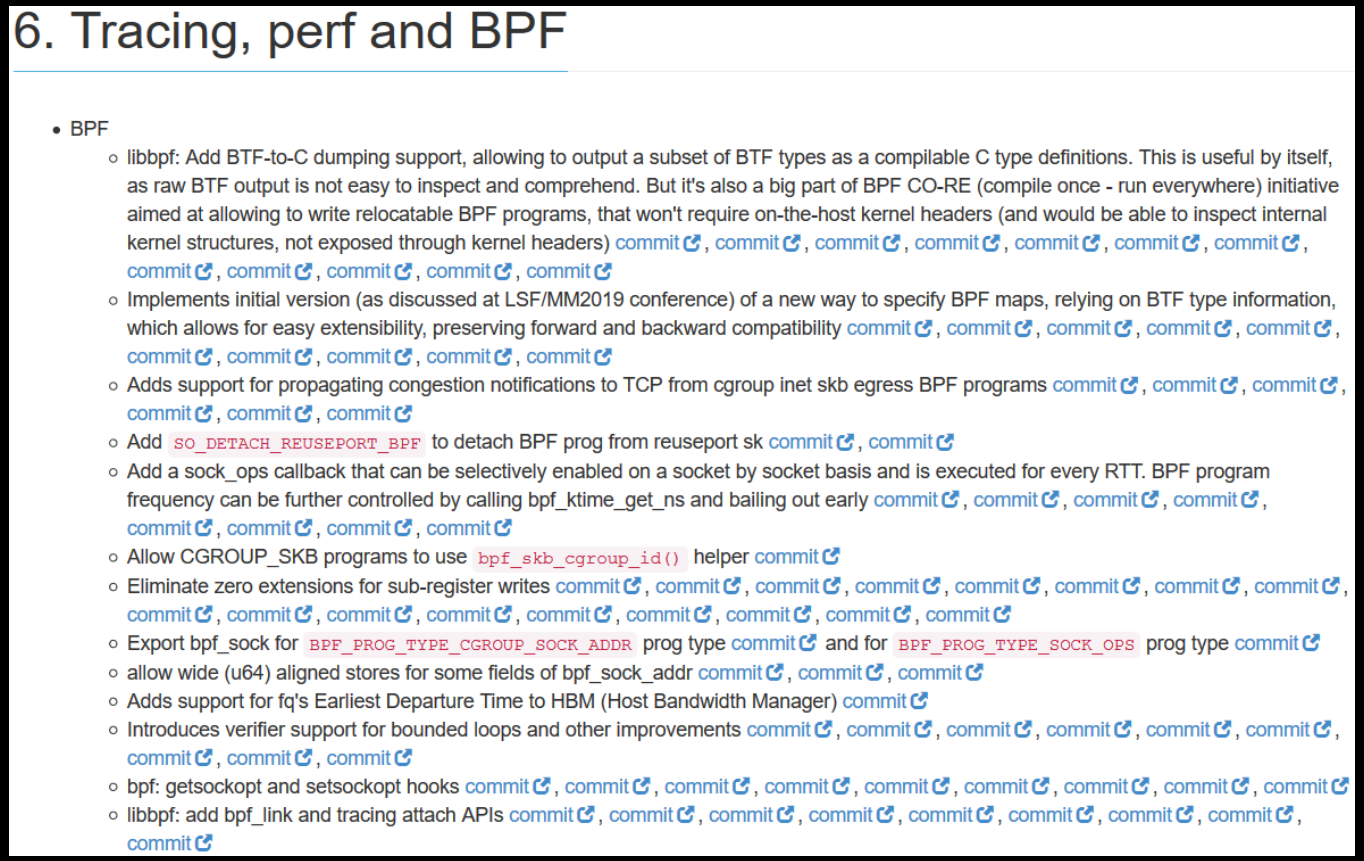
Workflow



Source: http://www.slideshare.net/vh21/meet-cutebetweenebpfandtracing

Internals

- **PIs refer to my presentation "eBPF in Action" at LC3 Beijing (on Jun 25, 2018)**
- **$KERNEL_SRC/Documentation/networking/filter.txt**
- **https://kernelnewbies.org/Linux_5.3**

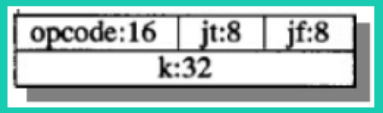
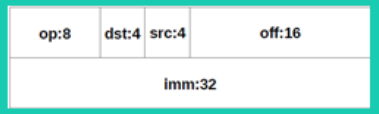


6. Tracing, perf and BPF

- BPF
  - libbpf: Add BTF-to-C dumping support, allowing to output a subset of BTF types as a compilable C type definitions. This is useful by itself, as raw BTF output is not easy to inspect and comprehend. But it's also a big part of BPF CO-RE (compile once - run everywhere) initiative aimed at allowing to write relocatable BPF programs, that won't require on-the-host kernel headers (and would be able to inspect internal kernel structures, not exposed through kernel headers) commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit
  - Implements initial version (as discussed at LSF/MM2019 conference) of a new way to specify BPF maps, relying on BTF type information, which allows for easy extensibility, preserving forward and backward compatibility commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit
  - Adds support for propagating congestion notifications to TCP from cgroup inet skb egress BPF programs commit, commit, commit, commit, commit, commit
  - Add `SO_DETACH_REUSEPORT_BPF` to detach BPF prog from reuseport sk commit, commit
  - Add a sock_ops callback that can be selectively enabled on a socket by socket basis and is executed for every RTT. BPF program frequency can be further controlled by calling bpf_ktime_get_ns and bailing out early commit, commit, commit, commit, commit, commit, commit, commit, commit
  - Allow CGROUP_SKB programs to use `bpf_skb_cgroup_id()` helper commit
  - Eliminate zero extensions for sub-register writes commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit, commit
  - Export bpf_sock for `BPF_PROG_TYPE_CGROUP_SOCK_ADDR` prog type commit and for `BPF_PROG_TYPE_SOCK_OPS` prog type commit
  - allow wide (u64) aligned stores for some fields of bpf_sock_addr commit, commit, commit
  - Adds support for fq's Earliest Departure Time to HBM (Host Bandwidth Manager) commit
  - Introduces verifier support for bounded loops and other improvements commit, commit, commit, commit, commit, commit, commit, commit, commit, commit
  - bpf: getsockopt and setsockopt hooks commit, commit, commit, commit, commit, commit, commit, commit, commit, commit
  - libbpf: add bpf_link and tracing attach APIs commit, commit, commit, commit, commit, commit, commit, commit, commit, commit

- **...**

Comparison

	cBPF	eBPF
Register	Two 32 bit registers: A: accumulator X: indexing	Eleven 64 bit registers: R0: return value/exit value R1-R5: arguments R6-R9: callee saved registers R10: read-only frame pointer
Instruction	~30 	~90 
JIT	Support	Support (better mapping with newer architectures for JITing)
Toolchain	GCC, tools/net	LLVM eBPF backend
Platform	x86_64, ARM, ARM64, SPARC, PowerPC, MIPS and s390	x86-64, aarch64, s390x...

System Call		`#include <linux/bpf.h>`
		`int bpf(int cmd, union bpf_attr *attr, unsigned int size);`
		(CALL, MAP, LOAD...)

Application	tcpdump... apply for seccomp filters, traffic control…	DDoS Mitigation, Intrusion Detection, Container Security, SDN Configuration, Observability...

- bpf() system call
  http://www.man7.org/linux/man-pages/man2/bpf.2.html

1.3 XDP (eXpress Data Path)

- The https://www.iovisor.org/technology/xdp
- https://lwn.net/Articles/708087/ //Debating the value of XDP
- Generic hook

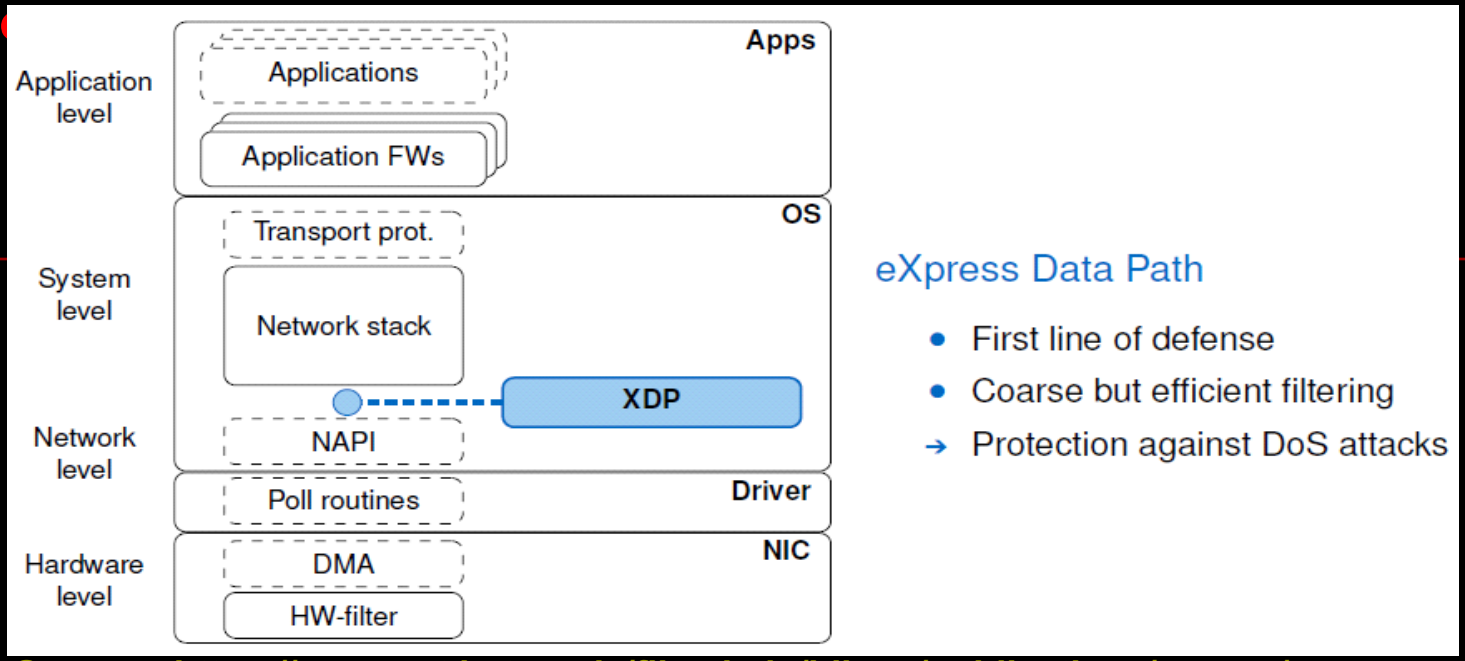
XDP is a further step in evolution and enables to run a specific flavor of BPF programs from the network driver with direct access to the packet's DMA buffer. This is, by definition, the earliest possible point in the software stack, where programs can be attached to in order to allow for a programmable, high performance packet processor in the Linux kernel networking data path.

Source: https://github.com/cilium/cilium

- Native XDP & Generic XDP

- XDP requires implementation in each driver
  - Need to choose XDP-supported driver
  - Not so handy
- Generic XDP allows you to use XDP on any driver (kernel 4.12)
  - XDP implemented in network stack
    - Convert skb to xdp buffer
  - Not as fast as native (non-generic) XDP
    - Need skb allocation at drivers
    - Packet buffer copy to meet XDP requirements
  - Good for functionality testing, etc.

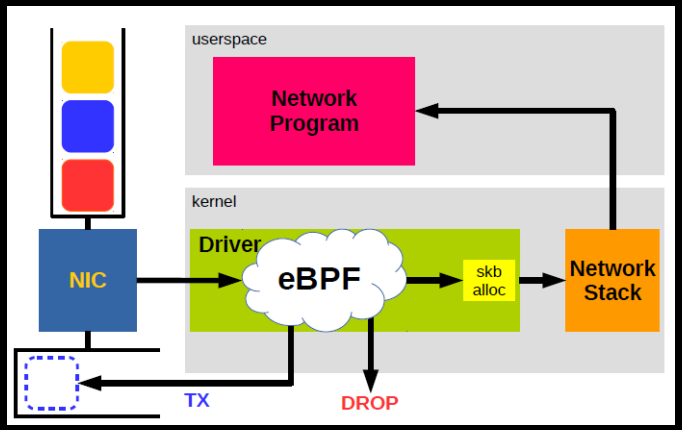
Source: "Veth XDP--XDP for containers", Toshiaki Makita & William Tu, NetDev 2019



eXpress Data Path

- First line of defense
- Coarse but efficient filtering
- → Protection against DoS attacks

Source: https://www.net.in.tum.de/fileadmin/bibtex/publications/papers/ITC30-Packet-Filtering-eBPF-XDP-slides.pdf



Source: https://www.slideshare.net/lcplcp1/xdp-and-ebpfmaps

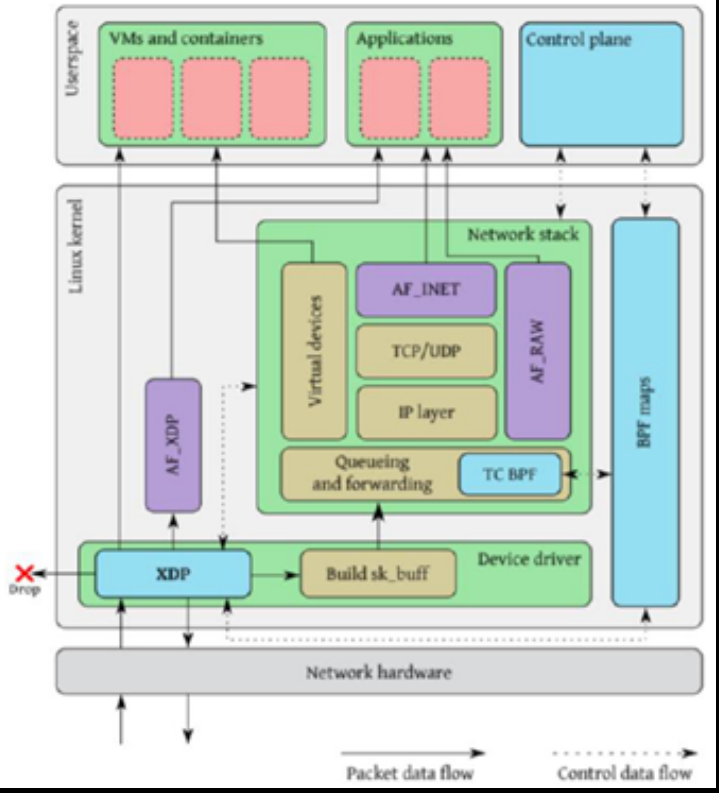
- eBPF trigger actions based on return codes
  - **XDP_DROP** - very fast drop by recycling
    - DDoS mitigation
  - **XDP_PASS** – pass possibly modified packet to network stack
    - Handle and pop new unknown encap protocols
  - **XDP_TX** – Transmit packet back out same interface
    - Facebook use it for load-balancing, and DDoS scrubber
  - **XDP_ABORTED** – also drop, but indicate error condition
    - Tracepoint: xdp_exception
  - **XDP_REDIRECT** – Transmit out other NICs
    - Very new (est.4.14), (plan also use for steering packets CPUs + sockets)

Source: http://people.netfilter.org/hawk/presentations/theCamp2017/theCamp2017_XDP_eBPF_technology_Jesper_Brouer.pdf

AF_XDP

- https://www.kernel.org/doc/html/latest/networking/af_xdp.html
- https://lwn.net/Articles/750293/
- https://lwn.net/Articles/750845/

- Overview
  - XDP's user space interface
  - Use XDP program to trigger Rx path for selected queue
  - Zero Copy from DMA buffers to user space with driver support
  - Copy mode for non-modified drivers
- Benefits
  - Performance boost
  - Support all Linux network devices



Source: "Integrating AF_XDP into DPDK", XIAO LONG YE, DPDK Summit China 2019

1.4 BPF development is "100% driven by use cases"

- https://lwn.net/Articles/801871/
- BPF Contributors

380 Daniel Borkmann (Cilium, Maintainer)
161 Alexei Starovoitov (Facebook, Maintainer)
160 Jakub Kicinski Netronome
110 John Fastabend (Cilium)
96 Yonghong Song (Facebook)
95 Martin KaFai Lau (Facebook)
94 Jesper Dangaard Brouer (Red Hat)
74 Quentin Monnet (Netronome)
45 Roman Gushchin (Facebook)
45 Andrey Ignatov (Facebook)

Top contributors of the total 186 contributors to BPF from January 2016 to November 2018.

Source: "BPF--Turning Linux into a Microservices-aware Operating System", Thomas Graf

2) Development
2.1 Toolchain
LLVM

- eBPF backend firstly introduced in LLVM 3.7 release
- http://llvm.org/docs/CodeGenerator.html#the-extended-berkeley-packet-filter-ebpf-backend
- (
  - Enabled by default with all major distributions
    - Registered targets: `llc --version`
    - `llc`'s BPF `-march` options: bpf, bpfeb, bpfel
    - `llc`'s BPF `-mcpu` options: generic, v1, v2, probe
  - Assembler output through `-S` supported
  - `llvm-objdump` for disassembler and code annotations (via DWARF)
  - Annotations correlate directly with kernel verifier log
  - Outputs ELF file with maps as relocation entries
    - Processed by BPF loaders (e.g. iproute2) and pushed into kernel

  Source: https://ossna2017.sched.com/event/BCsg/making-the-kernels-networking-data-path-programmable-with-bpf-and-xdp-daniel-borkmann-covalent
- $LLVM_SRC/lib/Target/BPF
- http://cilium.readthedocs.io/en/latest/bpf/
- LLVM 9.0 Released With Ability To Build The Linux x86_64

GCC (GCC support for eBPF is on the way)

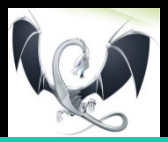
- https://www.phoronix.com/scan.php?page=news_item&px=GNU-Binutils-eBPF-Support
  //GNU Binutils Begins Landing eBPF Support
- https://www.phoronix.com/scan.php?page=news_item&px=GCC-10-eBPF-Backend-Plans
  //Oracle Is Aiming To Contribute An eBPF Backend To The GCC 10 Compiler
- https://www.phoronix.com/scan.php?page=news_item&px=Oracle-More-DTrace-Linux-eBPF
  //Oracle Is Working To Upstream More Of DTrace To The Linux Kernel & eBPF Implementation
- https://www.phoronix.com/scan.php?page=news_item&px=Oracle-GCC-10-eBPF-V2
  //2019-8-17::Oracle Continues Working On eBPF Support For GCC 10
- https://www.phoronix.com/scan.php?page=news_item&px=GCC-10-eBPF-Port-Lands
  //GCC 10 Lands The eBPF Port For Targeting The Linux In-Kernel VM

Status

- **Phase 1:** add eBPF target to the toolchain
  - `bpf-unknown-none`
  - binutils (upstream since May 2019)
  - GCC (upstream since September 2019)
- **Phase 2:** make the generated programs palatable for the kernel loaders and verifier, and **keep it that way**.
- **Phase 3:** provide development goodies for eBPF developers
  - GNU simulator
  - GDB
  - ...

Source: "eBPF support in the GNU Toolchain", Jose E. Marchesi (Oracle), Linux Plumbers Conference 2019

LLVM vs GCC



GPL v3	UIUC, MIT
Front-end: CC1 / CPP	Front-end: Clang
ld.bfd / ld.gold	lld / mclinker
gdb	lldb
as / objdump	MC layer
glibc	llvm-libc?
libstdc++	libc++
libsupc++	libc++abi
libgcc	libcompiler-rt
libgccjit	libLLVMMCJIT
. . .	ORC JIT, Coroutines, Clangd, libclc, Falcon...

- http://lld.llvm.org/
- https://llvm.org/docs/Proposals/LLVMLibC.html

2.2 Debugging

- https://www.netronome.com/documents/143/UG_Getting_Started_with_eBPF_Offload.pdf
- int bpf(int cmd, union bpf_attr *attr, unsigned int size);

log_level

- 0: No debug output.
- 1: Debug information from the verifier (all instructions).
- 2: More information: add all register states after each instruction.

- llvm-objdump, llvm-mc…

bpftool

- $KERNEL_SRC/tools/bpf /bpftool

```
bpftool
├── bash-completion
├── btf.c
├── btf_dumper.c
├── cfg.c
├── cfg.h
├── cgroup.c
├── common.c
├── Documentation
├── feature.c
├── jit_disasm.c
├── json_writer.c
├── json_writer.h
├── main.c
├── main.h
├── Makefile
├── map.c
├── map_perf_ring.c
├── net.c
├── netlink_dumper.c
├── netlink_dumper.h
├── perf.c
├── prog.c
├── tracelog.c
├── xlated_dumper.c
└── xlated_dumper.h
```

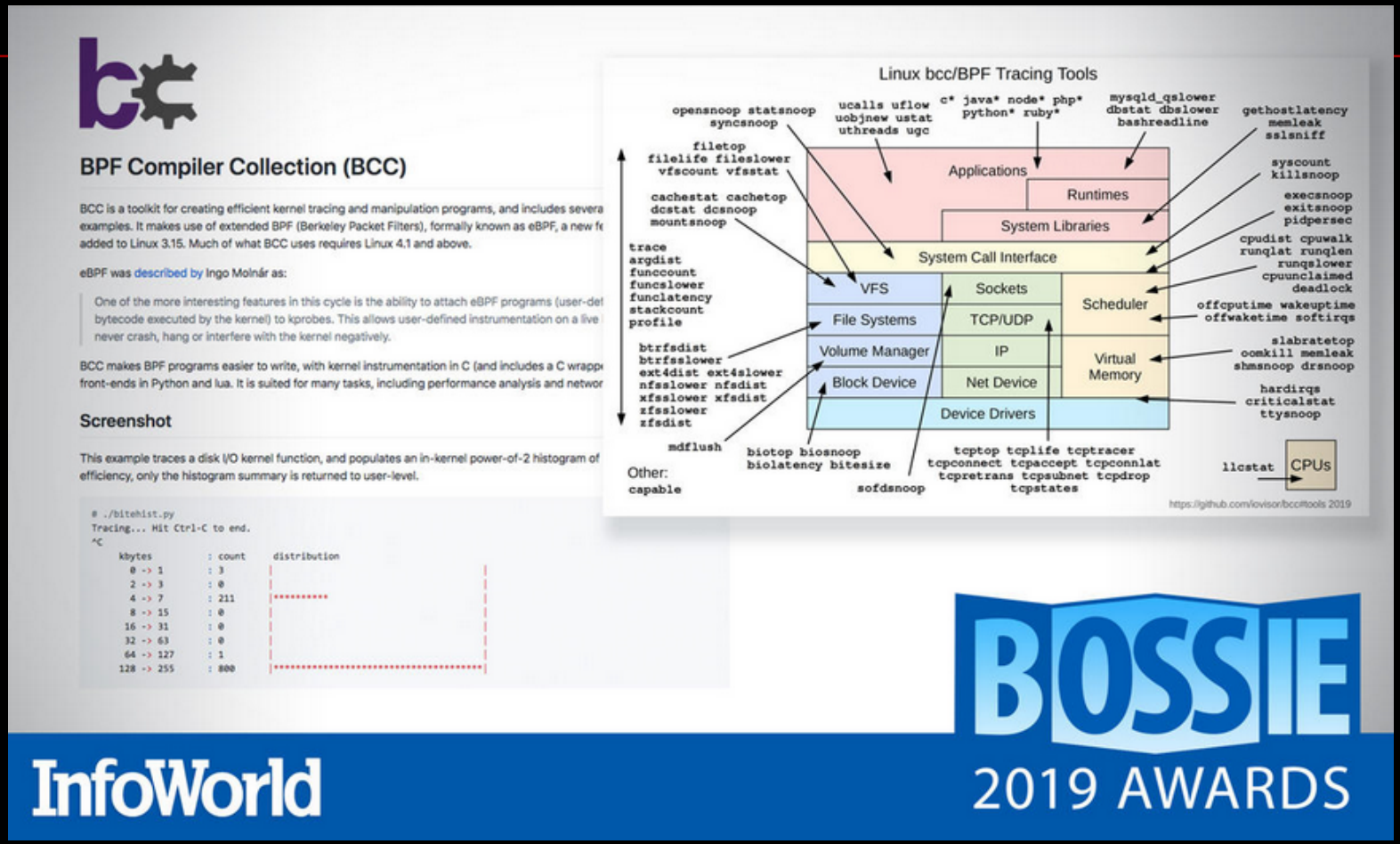
```
# bpftool prog show
1337: sched_cls  name cls_entry  tag e202124da7c84e89
      loaded_at Mar 08/19:53  uid 0
      xlated 304B  not jited  memlock 4096B
```

```
# bpftool prog dump xlated id 1337
   0: (71) r6 = *(u8 *)(r1 +142)
   1: (54) (u32) r6 &= (u32) 1
   2: (15) if r6 == 0x0 goto pc+7
   3: (bf) r6 = r1
   [...]
  37: (95) exit
```

```
# bpftool map
1234: array  name ch_rings  flags 0x0
      key 4B  value 4B  max_entries 7860  memlock 65536B

# bpftool map dump id 1234
key: 00 00 00 00  value: 00 00 00 00
key: 01 00 00 00  value: 00 00 00 00
key: 02 00 00 00  value: 00 00 00 00
key: 03 00 00 00  value: 00 00 00 00
[...]
Found 7860 elements
```

2.3 BCC (BPF Compiler Collection)

- https://www.infoworld.com/article/3444198/the-best-open-source-software-of-2019.html



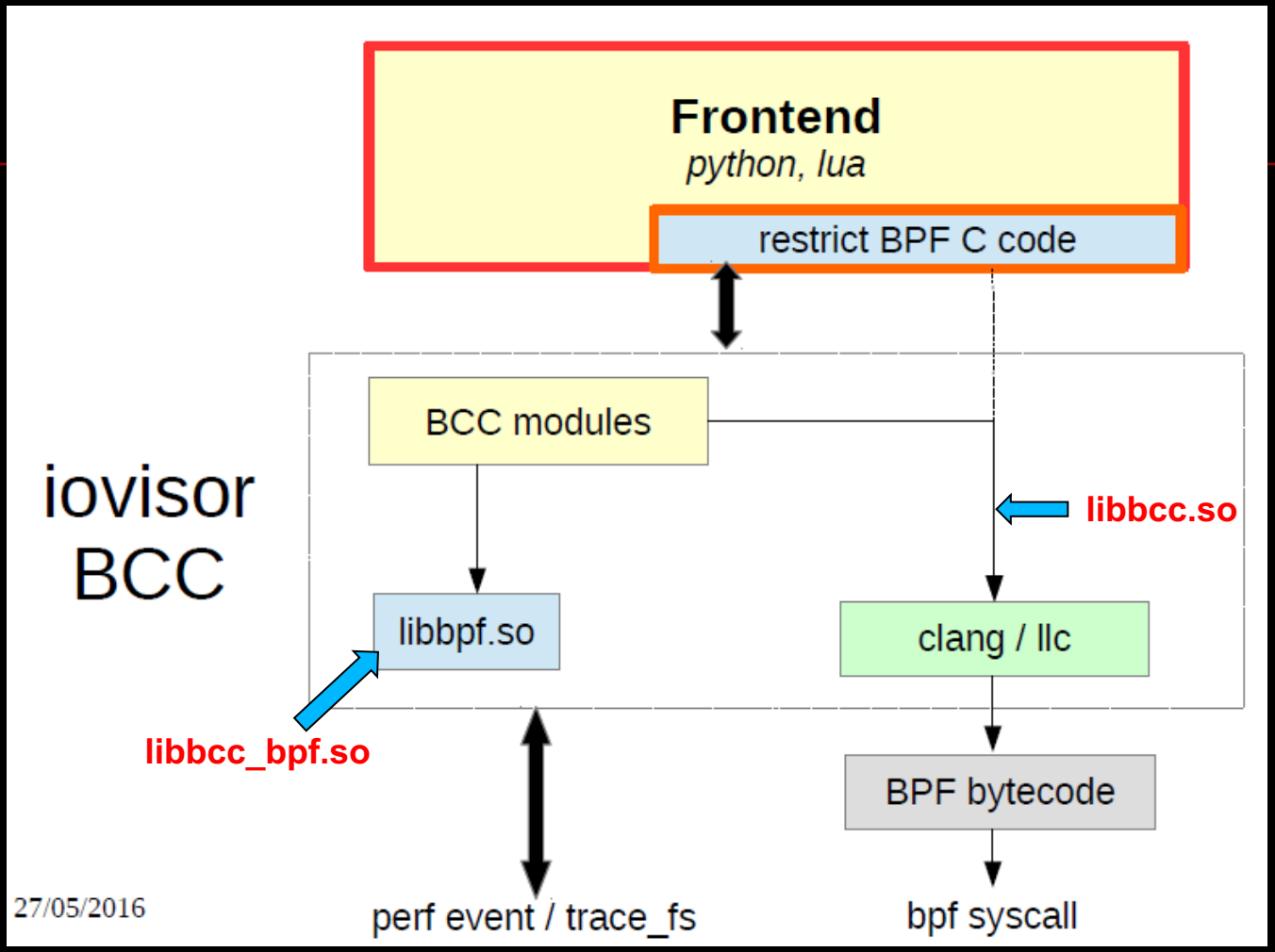
What is it

- https://github.com/iovisor/bcc/

a toolkit with Python/Lua frontend for compiling, loading, and executing BPF programs, which allows user-defined instrumentation on a live kernel image:

- compile BPF program from C source
- attach BPF program to kprobe/uprobe/tracepoint/USDT/socket
- poll data from BPF program
- framework for building new tools or one-off scripts
- contains a P4 compiler for BPF targets
- additional projects to support Go, Rust, and DTrace-style frontend
- …

Architecture

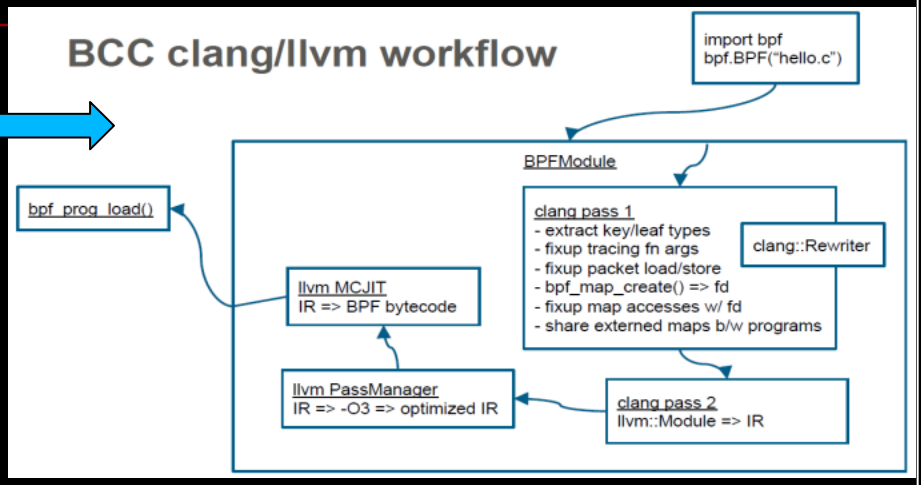


Source: http://www.slideshare.net/vh21/meet-cutebetweenebpfandtracing

A Sample

- https://lwn.net/Articles/747640/ //Some advanced BCC topics

```
#!/usr/bin/env python

from bcc import BPF
from time import sleep

program = """
    BPF_HASH(callers, u64, unsigned long);

    TRACEPOINT_PROBE(kmem, kmalloc) {
        u64 ip = args->call_site;
        unsigned long *count;
        unsigned long c = 1;

        count = callers.lookup((u64 *)&ip);
        if (count != 0)
            c += *count;

        callers.update(&ip, &c);

        return 0;
    }
"""
b = BPF(text=program)

while True:
    try:
        sleep(1)
        for k,v in sorted(b["callers"].items()):
            print ("%s %u" % (b.ksym(k.value), v.value))
        print
    except KeyboardInterrupt:
        exit()
```



Source: http://linuxplumbersconf.org/2015/ocw//system/presentations/3249/original/bpf_llvm_2015aug19.pdf

The output from this little program looks like:

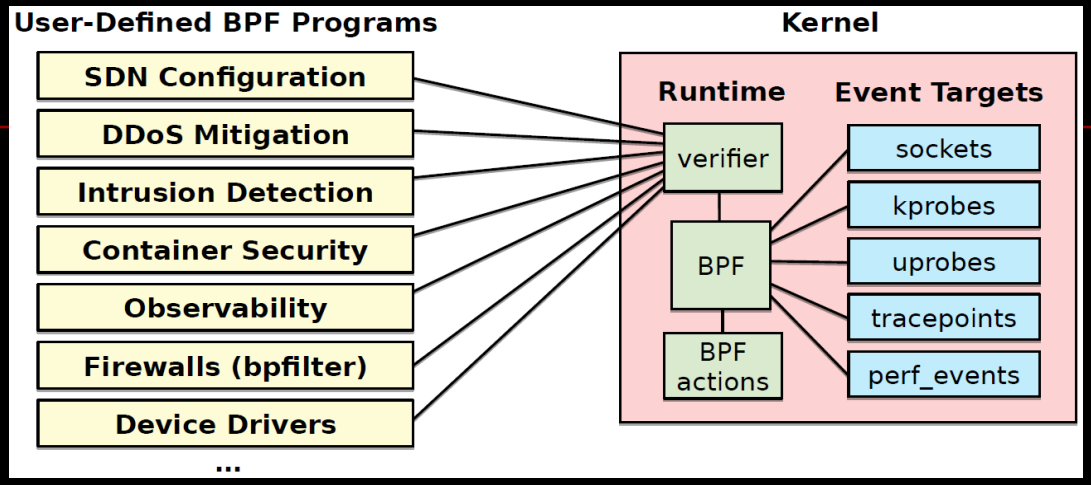
```
# ./example.py
i915_sw_fence_await_dma_fence 4
intel_crtc_duplicate_state 4
SyS_memfd_create 1
drm_atomic_state_init 4
sg_kmalloc 7
intel_atomic_state_alloc 4
seq_open 504
SyS_bpf 22
```

Development

- https://github.com/iovisor/bcc/blob/master/docs/tutorial.md
- https://github.com/iovisor/bcc/blob/master/docs/reference_guide.md
- https://github.com/iovisor/bcc/blob/master/docs/tutorial_bcc_python_developer.md
- …

3) Application

Overview



Source: "eBPF Perf Tools 2019", Brendan Gregg, SCaLE 17X

Netflix
Performance profiling and tracing

Facebook
eBPF-based load balancer with DDoS

Sysdig
eBPF instrumentation for high performance system calls tracing

Cloudflare
DDoS and Observability

Weaveworks
Trace TCP events

Cilium
Powerful and efficient networking, security and load-balancing at L3-L7.

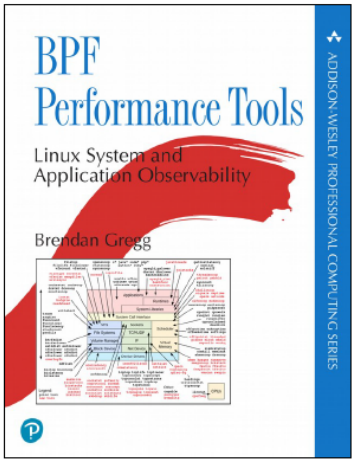
AWS Firecracker
Using Seccomp BPF to restrict system calls.

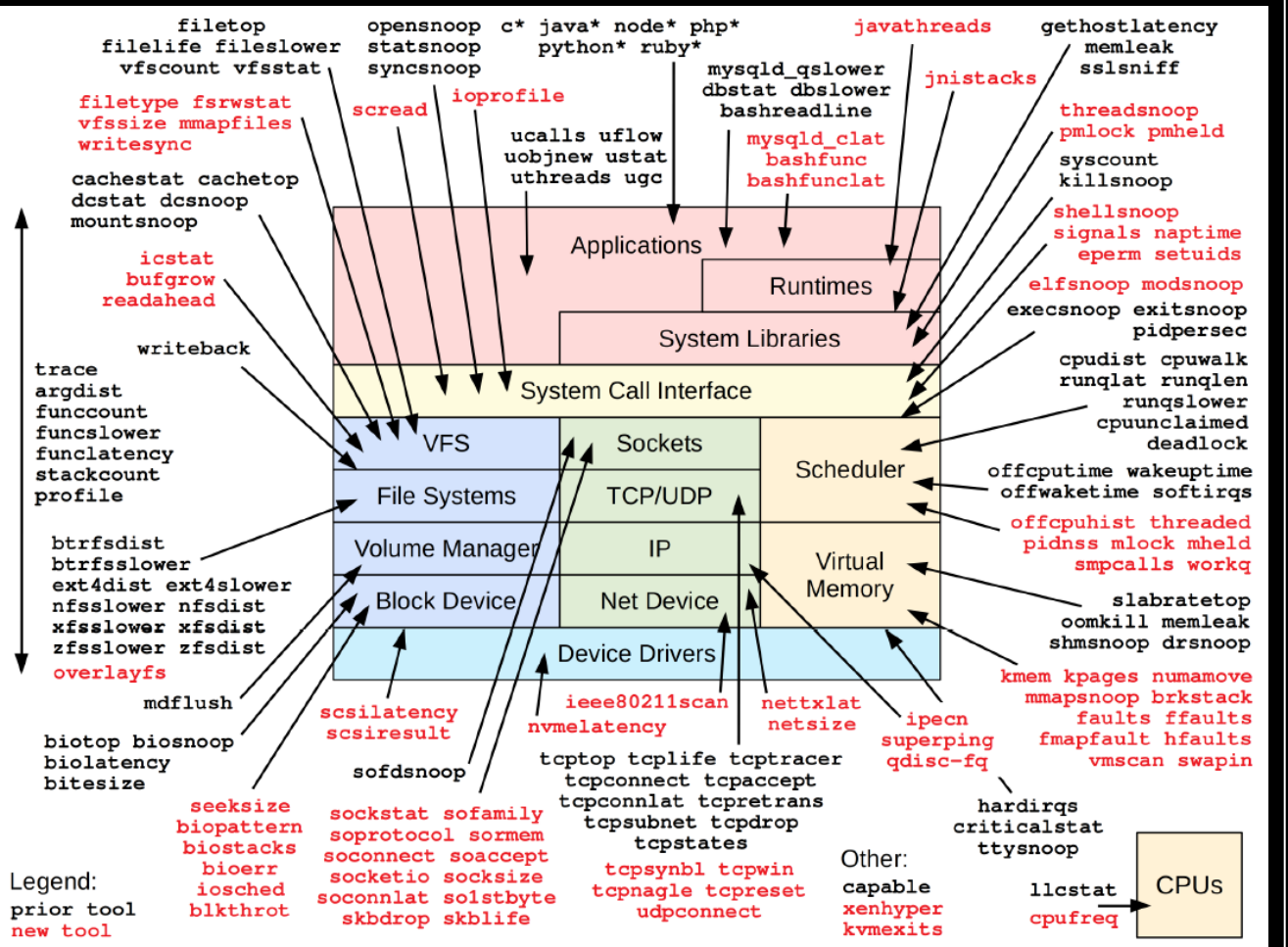
Redhat
RHEL 7.6 enables extended eBPF in-kernel VM

Source: "Back to the future with eBPF", Beatriz Martínez Rubio, KubeCon Europe 2019

Instrumentation

BPF Perf Tools





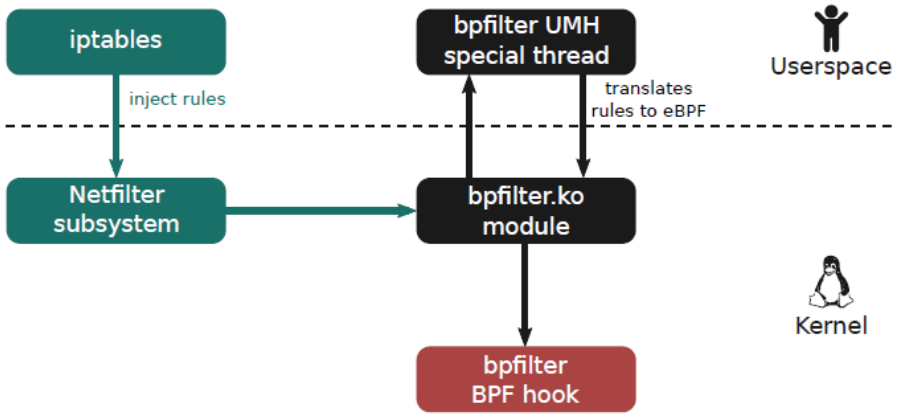
Source: “BPF Tracing Tools”, Brendan Gregg, Linux Plumbers Conference 2019

…

Security

- https://lwn.net/Articles/749108/ //Designing ELF modules
- https://lwn.net/Articles/755919/ //Bpfilter (and user-mode blobs)
- **bpfilter**: new back-end for iptables in Linux, based on BPF

  - The `iptables` binary is left untouched
  - Rules are translated into a BPF program
  - Uses a special kernel module launching an ELF executable in a special thread in user space, for rule translation
  - Also: proposal for nf_tables to BPF translation on top of bpfilter



Motivation:

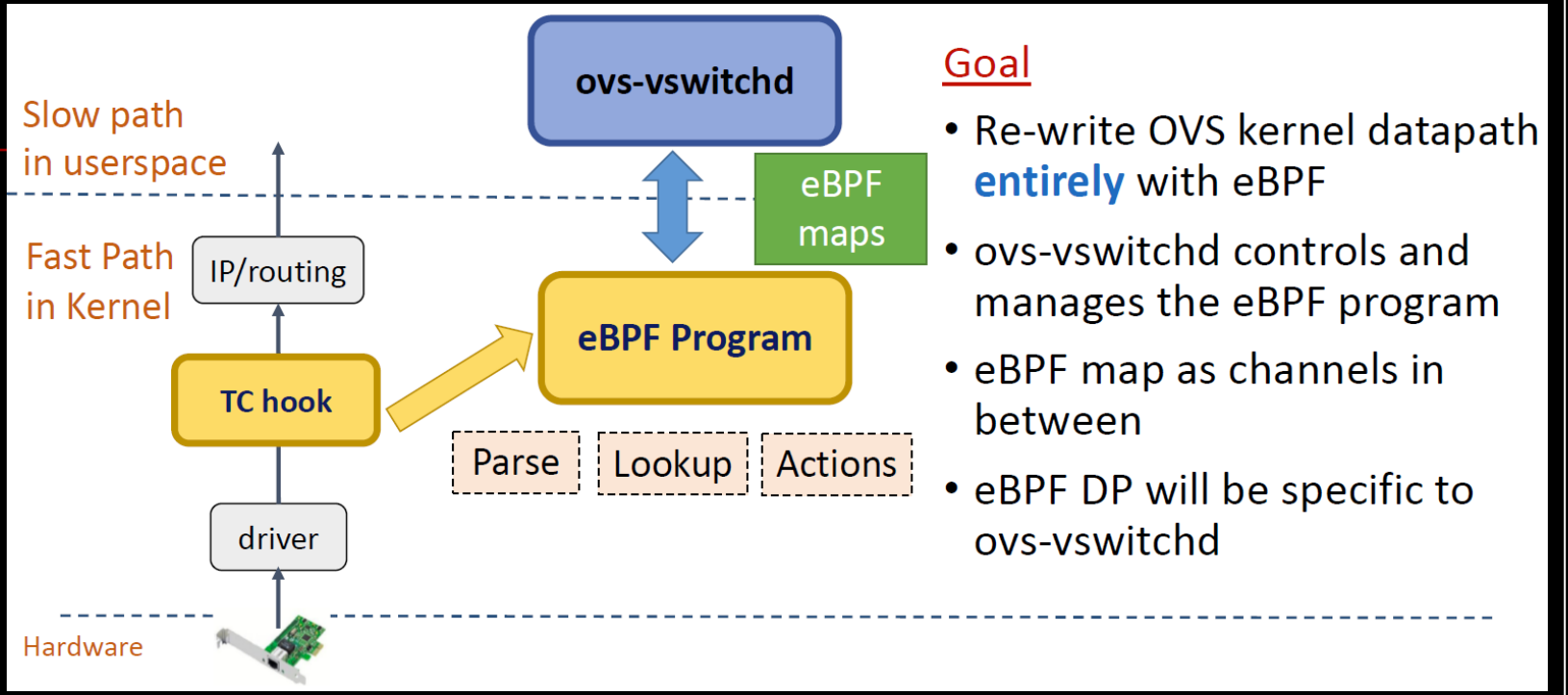
- Reuse rules from iptables
- Improve performance (JIT, offloads)

Source: "Unifying network filtering rules for the Linux kernel with eBPF"
Quentin Monnet(Netronome Systems Inc), Fosdem 2019

...

Virtualization

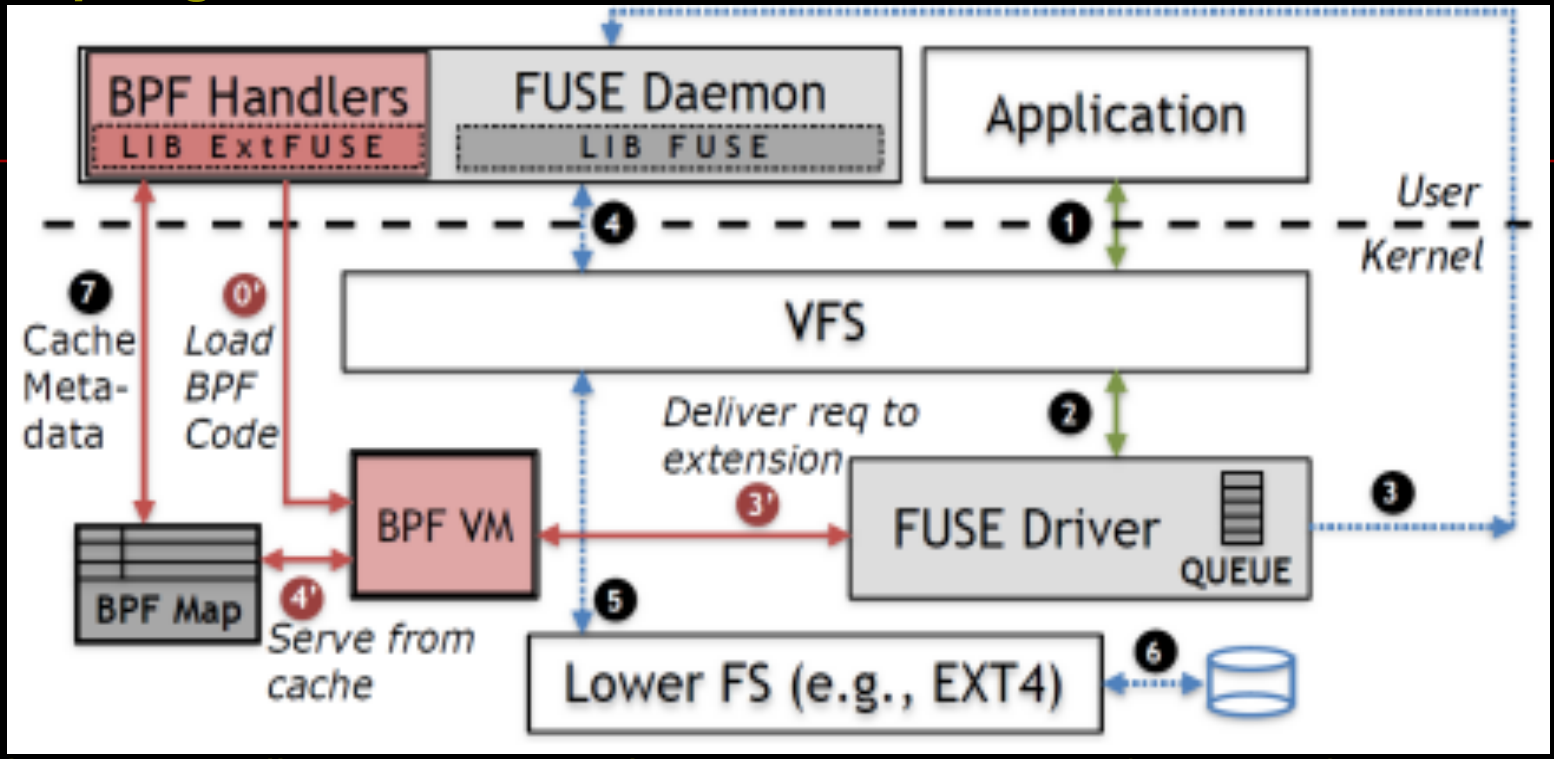
- https://github.com/williamtu/ovs-ebpf



Source: "Bringing the Power of eBPF to Open vSwitch", LPC 2018

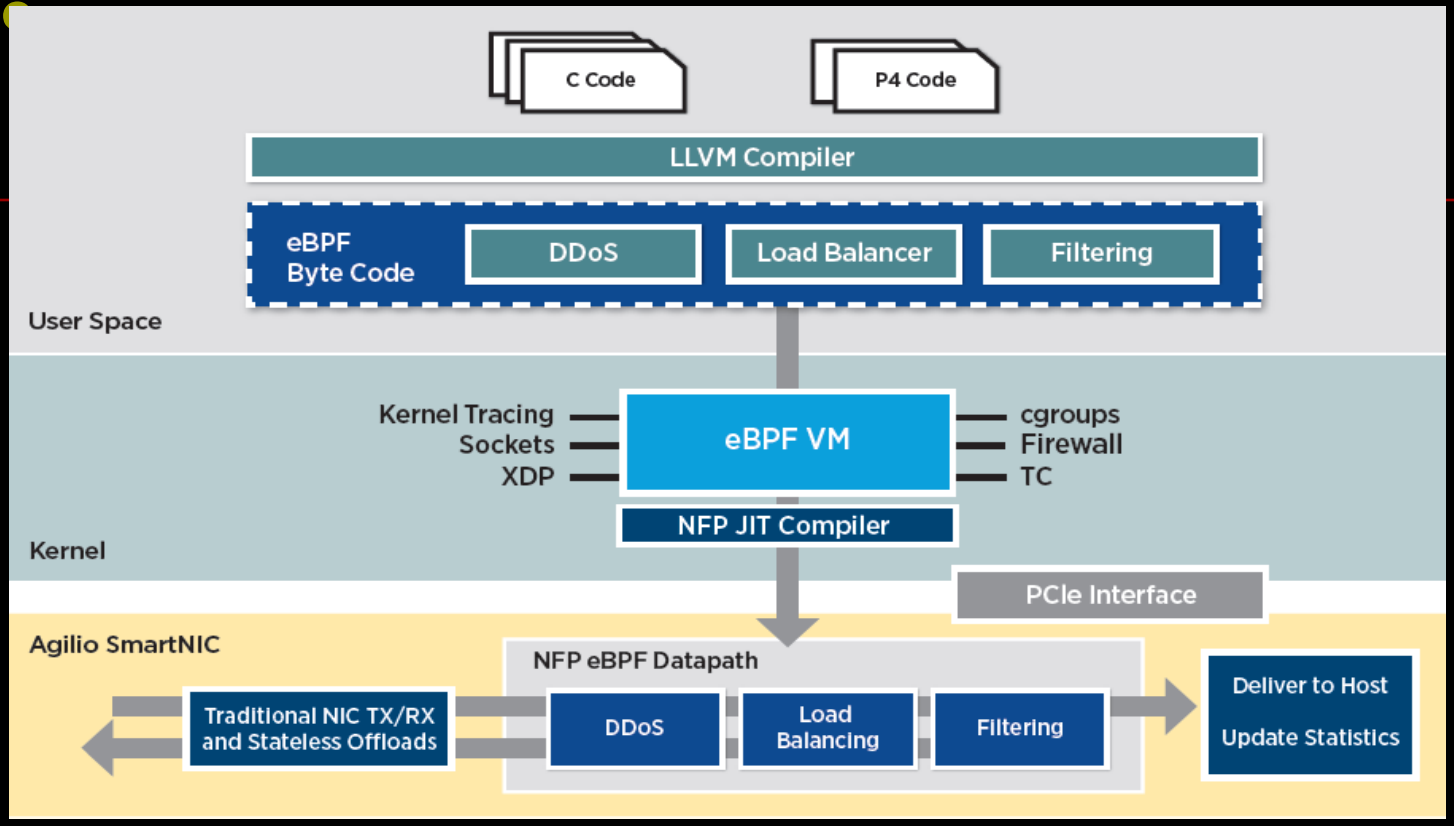
Storage

- **https://github.com/extfuse**



Source: https://www.phoronix.com/scan.php?page=news_item&px=ExtFUSE-Faster-FUSE-eBPF

SmartNIC



Source: https://www.netronome.com/m/documents/PB_Agilio-eBPF.pdf

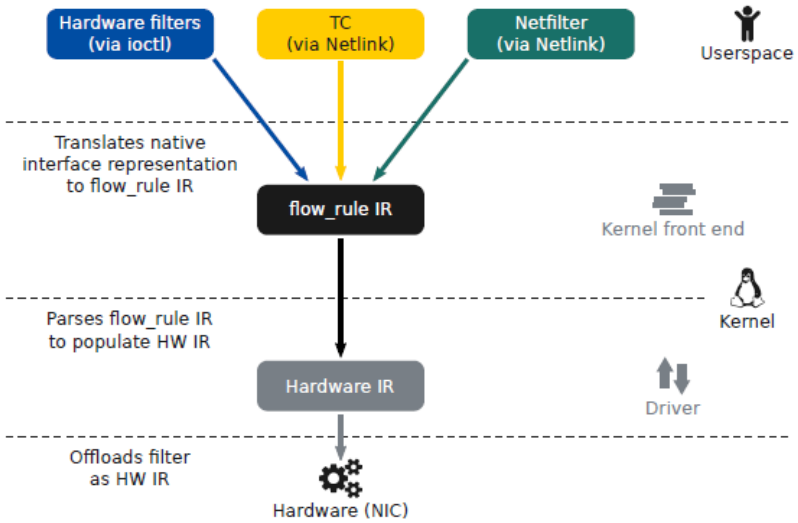
- https://www.nextplatform.com/2018/08/06/living-in-the-smartnic-future/

Unified Rules Handling

- https://lwn.net/Articles/775046/ //add flow_rule infrastructure
-

Work in progress from Pablo Neira Ayuso—No BPF in this one

- Intermediate representation for ACL hardware offloads
- Based on Linux flow dissector infrastructure and TC actions
- Can be used by different front-ends such as HW filters, TC, Netfilter



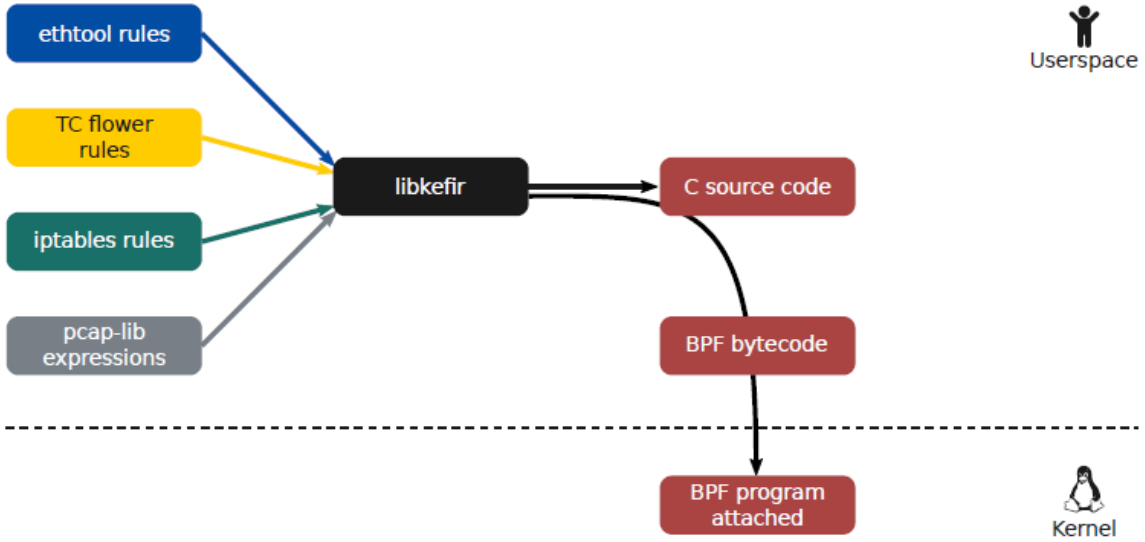
Motivation:

- Unified IR passed to the driver: avoid having one parser for each ACL front-end
- Stop exposing TC front-end details to drivers (easier to add features to TC)

Source: "Unifying network filtering rules for the Linux kernel with eBPF"
Quentin Monnet(Netronome Systems Inc), Fosdem 2019

libkefir: KErnel FIltering Rules: Work in progress @ Netronome

- Turn simple ACL rules into hackable BPF programs
- Motivation similar to bpfilter: reuse rules, with improved performance
- But do not try to handle all cases
- And give BPF-compatible C source code to users, so they can hack it
- Comes as a library, for inclusion in other projects



Sorry, not published yet!

Source: "Unifying network filtering rules for the Linux kernel with eBPF" Quentin Monnet(Netronome Systems Inc), Fosdem 2019

4) Summary

- **Evaluation**

"eBPF is Linux's new superpower"

Gaurav Gupta

"eBPF does to Linux what JavaScript does to HTML"

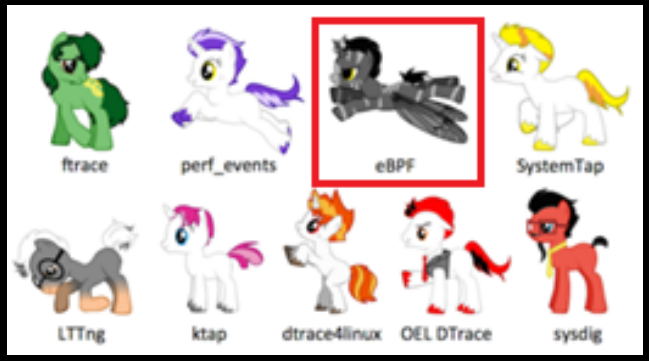
Brendan Gregg

"Run code in the kernel without having to write a kernel module"

Liz Rice

Source: ebpfbasics-190611051559.pdf

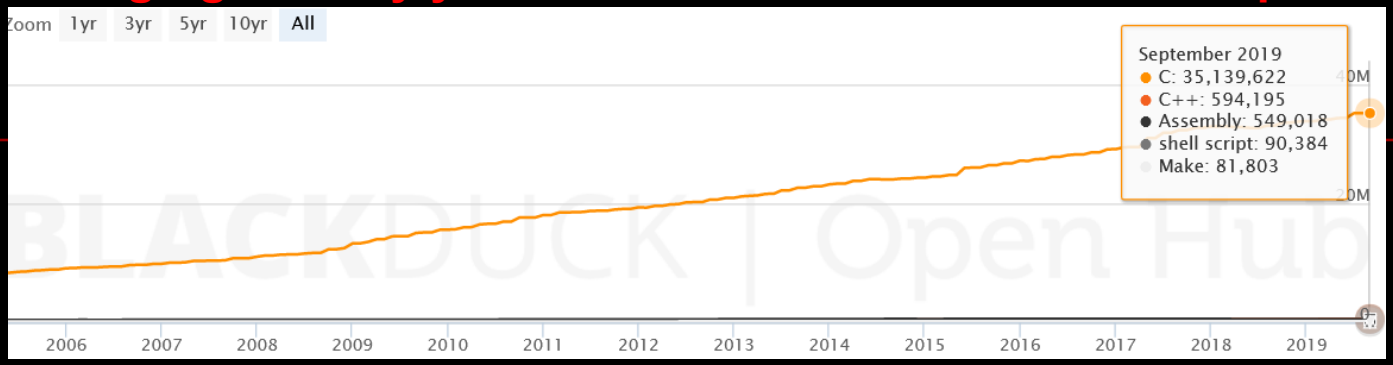
- **Kernel Space & User Space Instrumentation**



- **Dynamically extend Kernel functionalities at runtime**

- **Potential Polyglot VM**

  Changing the way you think about Linux Kernel development:



Language	Code Lines	Comment Lines	Comment Ratio	Blank Lines	Total Lines	Total Percentage
C	35,139,622	6,241,390	15.1%	6,333,153	47,714,165	95.7%
C++	594,195	262,892	30.7%	116,246	973,333	2.0%
Assembly	549,018	124,794	18.5%	96,032	769,844	1.5%

Source: https://www.openhub.net/p/linux/analyses/latest/languages_summary

- **The Next Linux Superpower:**

  Reconstructing nearly every aspect of Linux Networking and Security subsystem, and more other subsystems are on the way…

Pros & Cons

Pros

- Could replace lots of debugfs files
- No need kernel debug symbols
- Scalable for dynamic probing
- Lower performance impact than even perf events
- Security: sandboxing + verifier
- On-the-fly program generation
- …

Cons

- Up to 512 bytes stack
- Max 4096 instructions per program
- No more than 64 maps
- Performance of verifier needs to improve
- Security faults: Spectre vulnerability…
- …

eBPF is getting better, e.g., add bounded loops support in Linux 5.3 …

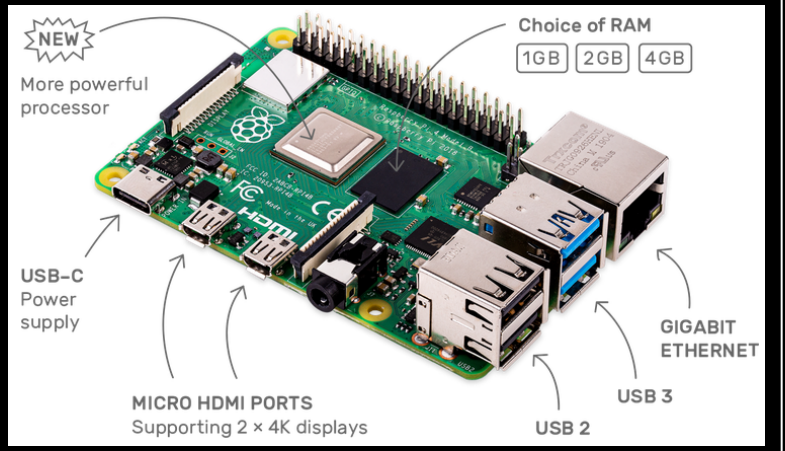
II. Testbed

1) Development Board

1.1 Raspberry Pi 4

- https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
- https://www.cnx-software.com/2019/06/24/raspberry-pi-4-vs-pi-3-what-are-the-differences/

Features/Specs	Raspberry Pi 4B	Raspberry Pi 3 B+
Release date	24th June 2019	14th March 2018
SoC	Broadcom BCM2711 quad-core Cortex-A72 @ 1.5 GHz	Broadcom BCM2837B0 quad-core Cortex-A53 @ 1.4 GHz
GPU	VideoCore VI with OpenGL ES 1.1, 2.0, 3.0	VideoCore IV with OpenGL ES 1.1, 2.0
Video Decode	H.265 4Kp60, H.264 1080p60	H.264 & MPEG-4 1080p30
Video Encode	H.264 1080p30	
Memory	1GB, 2GB, or 4GB LPDDR4	1GB LPDDR2
Storage	microSD card	
Video & Audio Output	2x micro HDMI ports up to 4Kp60 3.5mm AV port (composite + audio) MIPI DSI connector	1x HDMI 1.4 port up to 1080p60 3.5mm AV port (composite + audio) MIPI DSI connector
Camera	MIPI CSI connector	
Ethernet	Native Gigabit Ethernet	Gigabit Ethernet over USB (300 Mbps max.)
WiFi	Dual band 802.11 b/g/n/ac	
Bluetooth	Bluetooth 5.0 + BLE	Bluetooth 4.2 + BLE
USB	2x USB 3.0 + 2x USB 2.0	4x USB 2.0
Expansion	40-pin GPIO header	
Power Supply	5V via USB type-C up to 3A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT	5V via micro USB up to 2.5A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT
Dimensions	85×56 mm	
Default OS	Raspbian (after June 24, 2019)	Raspbian (after March 2018)
Price	$35 (1GB RAM), $45 (2GB RAM), $55 (4GB RAM)	$35 (1GB RAM)



2) Software Platform

2.1 Manjaro

- https://manjaro.org
- an open-source Linux distribution based on the Arch Linux OS

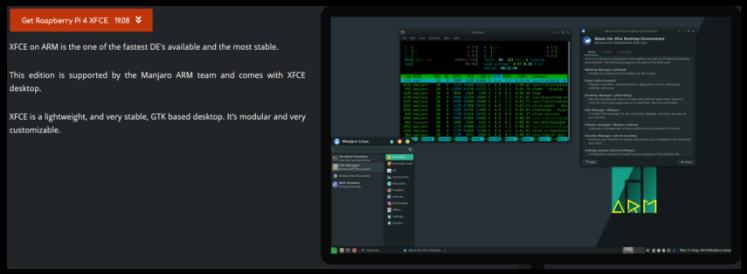
Rank	Distribution	H.P.D*
1	Mint	2880▼
2	Debian	1668▬
3	Ubuntu	1347▬
4	openSUSE	1234▲
5	elementary	1085▬
6	Manjaro	1039▼
7	Fedora	982▼
8	Zorin	914▬
9	CentOS	773▬
10	deepin	736▲

2016-->2019

https://www.distrowatch.com

Rank	Distribution	HPD*
1	MX Linux	4895
2	Manjaro	2597
3	Mint	2048
4	Debian	1543
5	Ubuntu	1398
6	elementary	1302
7	Solus	1087
8	Fedora	992
9	deepin	847
10	Zorin	830

- one of the first ARM64 Linux distribution that support RPi 4



- Advantages (from my point of view)
  - update packages quickly
  - group install

    https://www.archlinux.org/groups/
  - Python 3 is the default Python version now

    https://hg.python.org/peps/rev/76d43e52d978
  - much more stable than expected

  …

- https://www.howtogeek.com/430556/why-i-switched-from-ubuntu-to-manjaro-linux/
- https://wiki.archlinux.org/index.php/Pacman
- Preparation

upgrade original Kernel on Manjaro RPi4 from v4.19.x to v5.3.x with all the necessary options (like that for eBPF, Virtualization, Debugging, and so on) enabled

```
#
# eBPF
#
CONFIG_CGROUP_BPF=y
CONFIG_BPF=y
CONFIG_BPF_SYSCALL=y
CONFIG_BPF_JIT_ALWAYS_ON=y
CONFIG_IPV6_SEG6_BPF=y
CONFIG_NETFILTER_XT_MATCH_BPF=m
# CONFIG_BPFILTER is not set
CONFIG_NET_CLS_BPF=m
CONFIG_NET_ACT_BPF=m
CONFIG_BPF_JIT=y
CONFIG_BPF_STREAM_PARSER=y
CONFIG_LWTUNNEL_BPF=y
CONFIG_HAVE_EBPF_JIT=y
CONFIG_BPF_LIRC_MODE2=y
CONFIG_BPF_EVENTS=y
# CONFIG_TEST_BPF is not set
# XDP
CONFIG_XDP_SOCKETS=y
CONFIG_XDP_SOCKETS_DIAG=m
```

```
#
# Virtualization
#
CONFIG_VIRTUALIZATION=y
CONFIG_PARAVIRT=y
CONFIG_PARAVIRT_TIME_ACCOUNTING=y
CONFIG_VIRTIO=y
CONFIG_HAVE_VIRT_CPU_ACCOUNTING_GEN=y
CONFIG_BLK_MQ_VIRTIO=y
CONFIG_VIRTIO_VSOCKETS=m
CONFIG_VIRTIO_VSOCKETS_COMMON=m
CONFIG_NET_9P_VIRTIO=m
CONFIG_VIRTIO_BLK=m
CONFIG_SCSI_VIRTIO=m
CONFIG_VIRTIO_NET=m
CONFIG_VIRT_WIFI=m
CONFIG_VIRTIO_CONSOLE=m
CONFIG_HW_RANDOM_VIRTIO=m
CONFIG_DRM_VIRTIO_GPU=m
CONFIG_SND_VIRTUOSO=m
CONFIG_VIRTIO_MENU=y
CONFIG_VIRTIO_PCI=y
CONFIG_VIRTIO_PCI_LEGACY=y
CONFIG_VIRTIO_BALLOON=m
CONFIG_VIRTIO_INPUT=m
CONFIG_VIRTIO_MMIO=m
CONFIG_RPMSG_VIRTIO=m
CONFIG_CRYPTO_DEV_VIRTIO=m
CONFIG_DMA_VIRT_OPS=y
CONFIG_REGULATOR_VIRTUAL_CONSUMER=m
CONFIG_FB_VIRTUAL=m
CONFIG_DMA_VIRTUAL_CHANNELS=y
#CONFIG_VIRTIO_PMEM=m
#CONFIG_VIRTIO_IOMMU=m
```

```
#
# KVM
#
CONFIG_KVM=y
CONFIG_HAVE_KVM_IRQCHIP=y
CONFIG_HAVE_KVM_IRQFD=y
CONFIG_HAVE_KVM_IRQ_ROUTING=y
CONFIG_HAVE_KVM_EVENTFD=y
CONFIG_KVM_MMIO=y
CONFIG_HAVE_KVM_MSI=y
CONFIG_HAVE_KVM_CPU_RELAX_INTERCEPT=y
CONFIG_KVM_VFIO=y
CONFIG_HAVE_KVM_ARCH_TLB_FLUSH_ALL=y
CONFIG_KVM_GENERIC_DIRTYLOG_READ_PROTECT=y
CONFIG_HAVE_KVM_IRQ_BYPASS=y
CONFIG_HAVE_KVM_VCPU_RUN_PID_CHANGE=y
CONFIG_KVM_ARM_HOST=y
CONFIG_KVM_ARM_PMU=y
CONFIG_KVM_INDIRECT_VECTORS=y
```

```
#
# Network
#
CONFIG_IPV6=y
CONFIG_IP_VS_IPV6=y
CONFIG_IPV6_SEG6_LWTUNNEL=y
CONFIG_IPV6_SEG6_HMAC=y
CONFIG_IPV6_OPTIMISTIC_DAD=y
CONFIG_IPV6_MIP6=y
CONFIG_IPV6_GRE=m
CONFIG_IPV6_ILA=m
CONFIG_IPV6_VTI=m
#CONFIG_IPV6_FOU=m
#CONFIG_IPV6_FOU_TUNNEL=m
#CONFIG_AF_RXRPC_IPV6=y


#
# Miscs
#
CONFIG_KSM=y
CONFIG_PROC_EVENTS=y
#CONFIG_IKHEADERS=y
CONFIG_REMOTEPROC=m
CONFIG_REISERFS_PROC_INFO=y
CONFIG_PROC_CHILDREN=y
```

```
#
# Debug
#
CONFIG_ARCH_HAS_DEBUG_VIRTUAL=y
CONFIG_STRIP_ASM_SYMS=y
CONFIG_UNUSED_SYMBOLS=y
CONFIG_PM_DEBUG=y
CONFIG_CMA_DEBUGFS=y
CONFIG_L2TP_DEBUGFS=m
CONFIG_6LOWPAN_DEBUGFS=y
CONFIG_CFG80211_DEBUGFS=y
CONFIG_MAC80211_DEBUGFS=y
CONFIG_DEBUG_DEVRES=y
CONFIG_SCSI_DEBUG=m
CONFIG_DM_DEBUG=y
CONFIG_DM_DEBUG_BLOCK_MANAGER_LOCKING=y
CONFIG_ATH9K_DEBUGFS=y
CONFIG_ATH6KL_DEBUG=y
CONFIG_SUNRPC_DEBUG=y
CONFIG_DLM_DEBUG=y
CONFIG_DYNAMIC_DEBUG=y
CONFIG_DEBUG_INFO=y
#CONFIG_DEBUG_INFO_DWARF4=y
CONFIG_DEBUG_SECTION_MISMATCH=y
CONFIG_DEBUG_RODATA_TEST=y
CONFIG_DEBUG_VM=y
CONFIG_DEBUG_SHIRQ=y
CONFIG_DEBUG_LIST=y
CONFIG_ARM64_PTDUMP_DEBUGFS=y
CONFIG_DEBUG_WX=y
CONFIG_PROC_KCORE=y
CONFIG_PROC_VMCORE=y
CONFIG_PROC_VMCORE_DEVICE_DUMP=y
CONFIG_KEXEC=y
# CONFIG_KEXEC_FILE is not set
CONFIG_KEXEC_CORE=y
CONFIG_CRASH_DUMP=y
CONFIG_CRASH_CORE=y
CONFIG_CRASH=m
CONFIG_FTRACE_SYSCALLS=y
CONFIG_HWLAT_TRACER=y
```

- Pls refer to my presentation "Python for Linux Kernel Debugging" at PyCon China Hangzhou (Oct 19, 2019) for details

2.2 Ubuntu

- https://jamesachambers.com/raspberry-pi-ubuntu-server-18-04-2-installation-guide/
- https://ubuntu.com/blog/roadmap-for-official-support-for-the-raspberry-pi-4 //Nov 3, 2019

Raspberry Pi 4	Ubuntu Server 19.10	Ubuntu Server 18.04	Ubuntu Core
1GB			
2GB			
4GB	*Upcoming kernel fix*		

Current Next Future

2.3 Fedora

- official support for Raspberry Pi 4 may coming in Fedora 32☹
- developer friendly
- https://iot.fedoraproject.org

III. eBPF-based In-Kernel Service

1) Service function chaining

BPF Tail Calls

- http://man7.org/linux/man-pages/man7/bpf-helpers.7.html

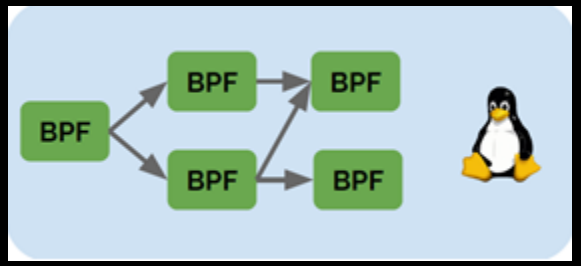
int bpf_tail_call(void *ctx, struct bpf_map *prog_array_map, u32 index)

Description

This special helper is used to trigger a "tail call", or in other words, to jump into another eBPF program. The same stack frame is used (but values on stack and in registers for the caller are not accessible to the callee). This mechanism allows for program chaining, either for raising the maximum number of available eBPF instructions, or to execute given programs in conditional blocks. For security reasons, there is an upper limit to the number of successive tail calls that can be performed.

Upon call of this helper, the program attempts to jump into a program referenced at index *index* in *prog_array_map*, a special map of type **BPF_MAP_TYPE_PROG_ARRAY**, and passes *ctx*, a pointer to the context.

If the call succeeds, the kernel immediately runs the first instruction of the new program. This is not a function call, and it never returns to the previous program. If the call fails, then the helper has no effect, and the caller continues to run its subsequent instructions. A call can fail if the destination program for the jump does not exist (i.e. *index* is superior to the number of entries in *prog_array_map*), or if the maximum number of tail calls has been reached for this chain of programs. This limit is defined in the kernel by the macro **MAX_TAIL_CALL_CNT** (not accessible to user space), which is currently set to 32.



2) Polycube

- https://github.com/polycube-network/polycube

- **eBPF/XDP-based software framework for fast network services running in the Linux kernel**

`Polycube` is an **open source** software framework for Linux that enables the creation of **virtual networks** and provides **fast** and **lightweight network functions**, such as *bridge*, *router*, *nat*, *load balancer*, *firewall*, *DDoS mitigator*, and more.

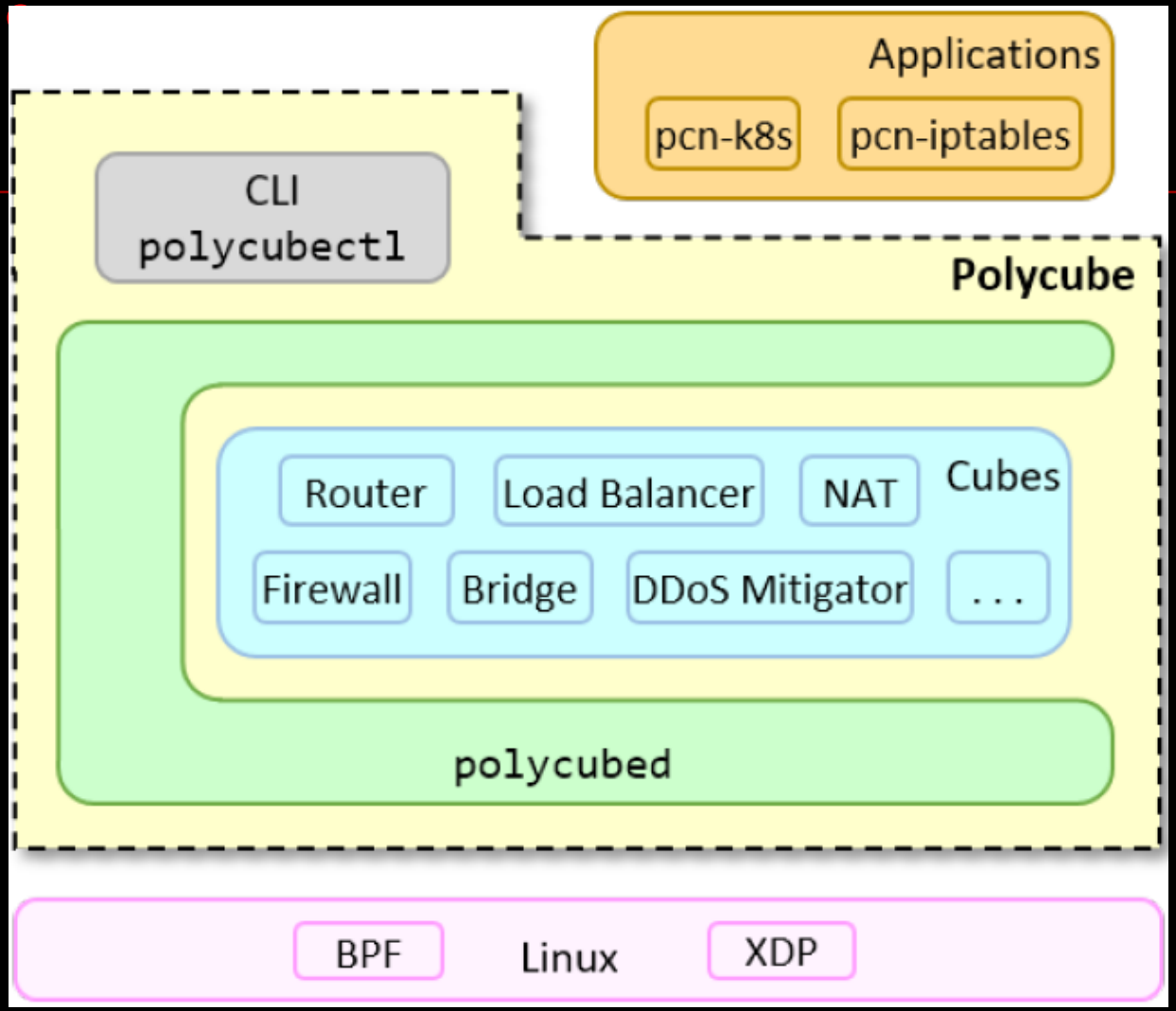
Within each virtual network, individual network functions can be composed to build arbitrary **service chains** and provide custom network connectivity to **namespaces**, **containers**, **virtual machines**, and **physical hosts**.

Virtual functions, called *cubes*, are extremely **efficient** because are based on the recent *BPF* and *XDP* Linux kernel technologies. In addition, cubes are easily **extensible** and **customizable**.

Polycube can control its entire virtual topology and all the network services with a simple and coherent command line, available through the *polycubectl* tool. A set of equivalent commands can be issued diretly to *polycubed*, the Polycube REST-based daemon, for better machine-to-machine interaction.

Polycube also provides two working **standalone applications** built up using this framework. *pcn-K8s* is a Polycube-based CNI plug-in for *Kubernetes*, which can handle the network of an entire data center. It also delivers better throughput as compared with some of the existing CNI plug-ins. *pcn-iptables* is a more efficient and scalable clone of the existing Linux *iptables*.

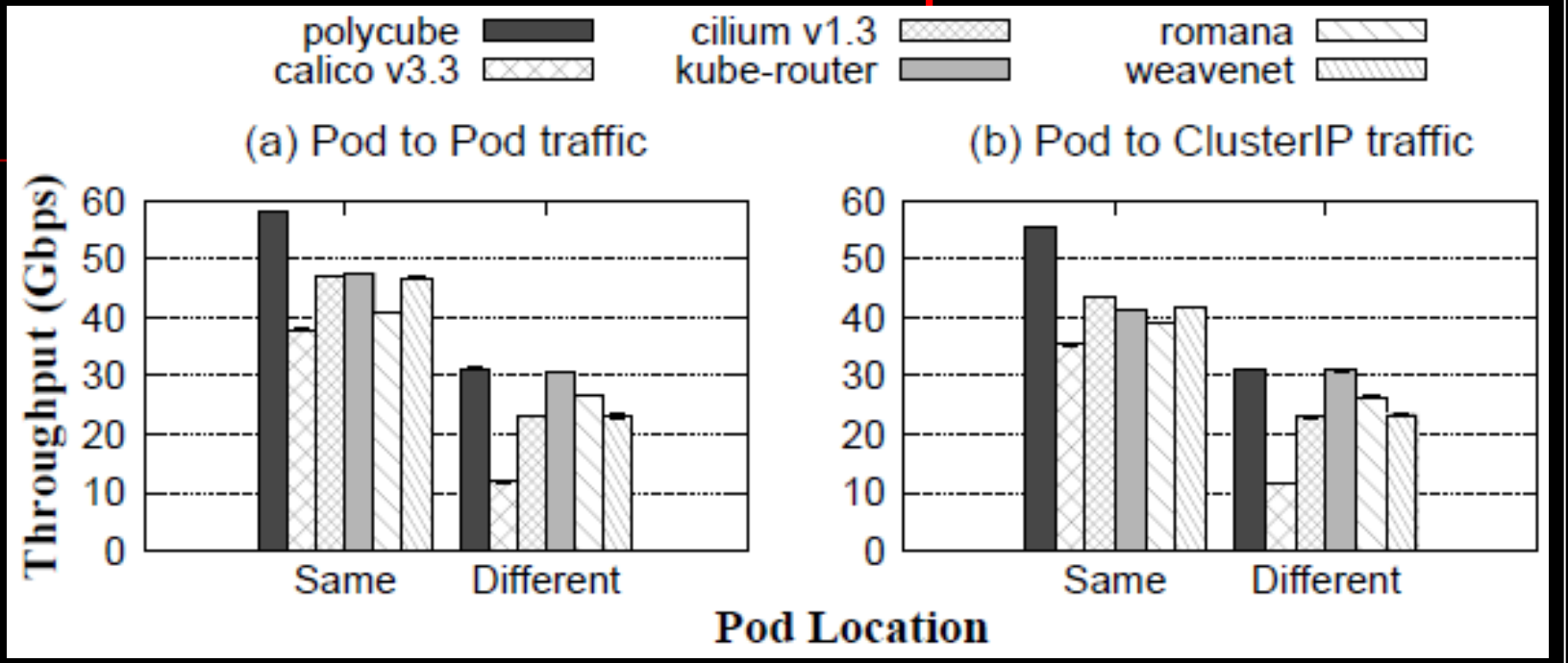
Architecure



Source: https://polycube-network.readthedocs.io/en/latest/

Performance

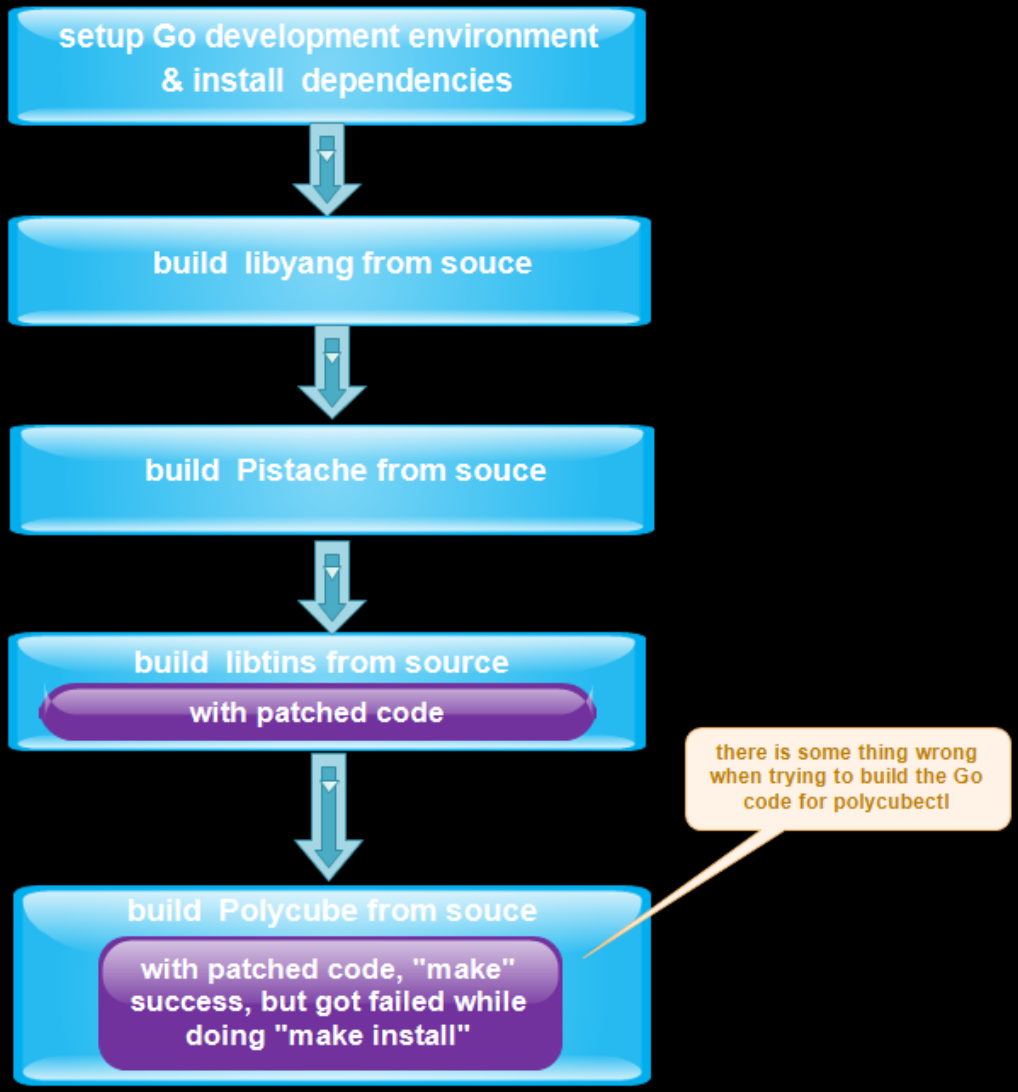
- **Performance of different k8s network providers**



Source: "A Service-Agnostic Software Framework for Fast and Efficient In-Kernel Network Services" Sebastiano Miano, Yunsong Lu etc, ANCS 2019

My Practice

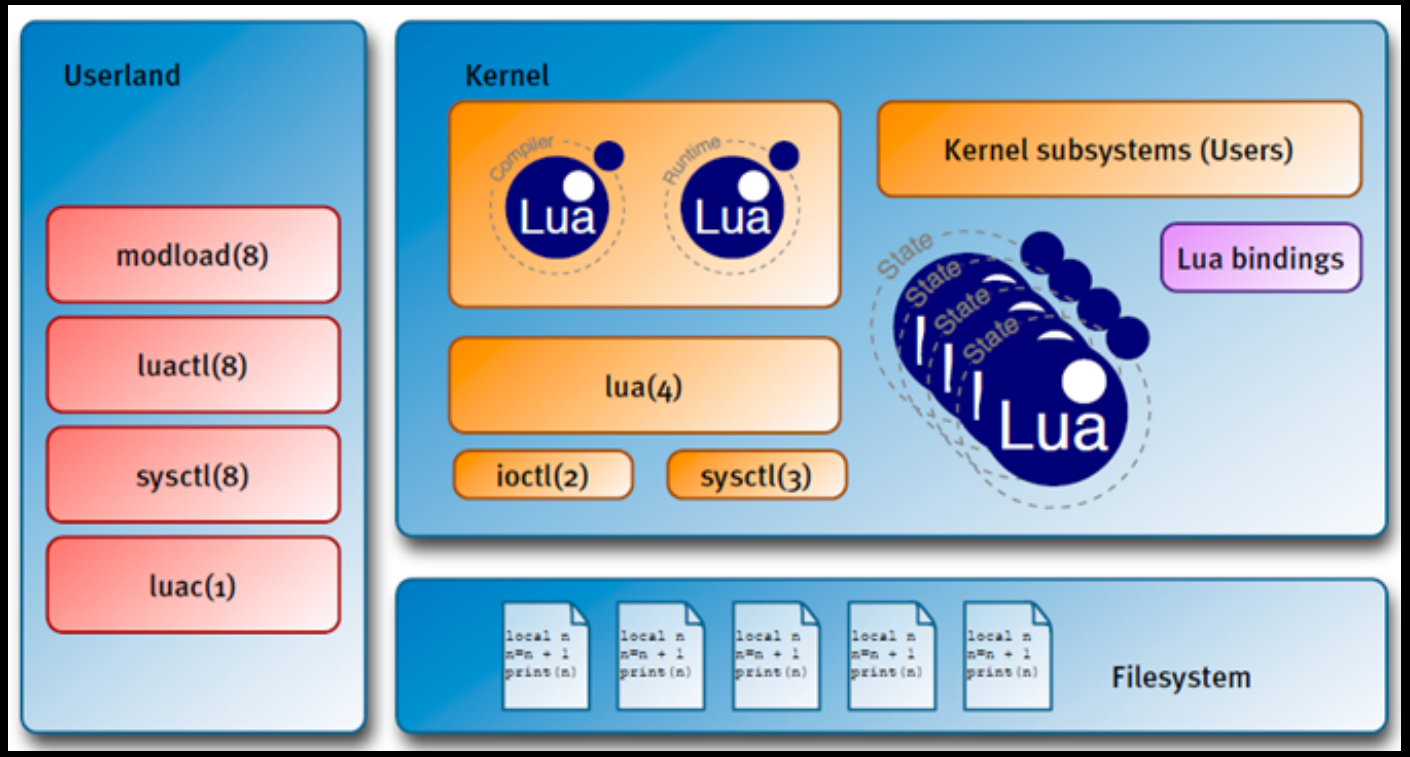
- building Polycube on ARM64



IV. Lua-based In-Kernal VM

1) NetBSD

- https://github.com/osandov/drgn
- NetBSD Kernel scripting with Lua
  be part of NetBSD 6 (Userland)
  be part of NetBSD 7 (Kernel)



Source: https://archive.fosdem.org/2013/schedule/event/lua_in_the_netbsd_kernel/

2) Linux

2.1 lunatik

- https://github.com/cujoai/lunatik
- a port of the Lua interpreter to the Linux kernel

2.2 lunatik-ng

- https://github.com/lunatik-ng/lunatik-ng
-

This repository contains the ongoing effort of porting the Lunatik Lua engine to current Linux kernels. There are a few differences between the original lunatik and lunatik-ng:

- Lunatik-ng works on x86_64
- It is memory-leak free
- It can be built as loadable modules
- A few interfaces to the kernel are provided by default:
  - `buffer` for allocating memory regions in kernel space
  - `crypto` which provides bindings to the SHA1 implementation in the kernel (a more advanced interface to the kernel which allows selection of the cipher is in the works) and the random number generator.
  - `printk` as a direct binding to the kernels `printk`
  - `type` and `gc_count` as bindings to parts of the default Lua library

2.3 KTap

- https://github.com/ktap/ktap

ktap is a new scripting dynamic tracing tool for Linux, it uses a scripting language and lets users trace the Linux kernel dynamically. ktap is designed to give operational insights with interoperability that allows users to tune, troubleshoot and extend the kernel and applications. It's similar to Linux Systemtap and Solaris Dtrace.

ktap has different design principles from Linux mainstream dynamic tracing language in that it's based on bytecode, so it doesn't depend upon GCC, doesn't require compiling kernel module for each script, safe to use in production environment, fulfilling the embedded ecosystem's tracing needs.

- https://github.com/ktap/ktap/blob/master/COPYRIGHT

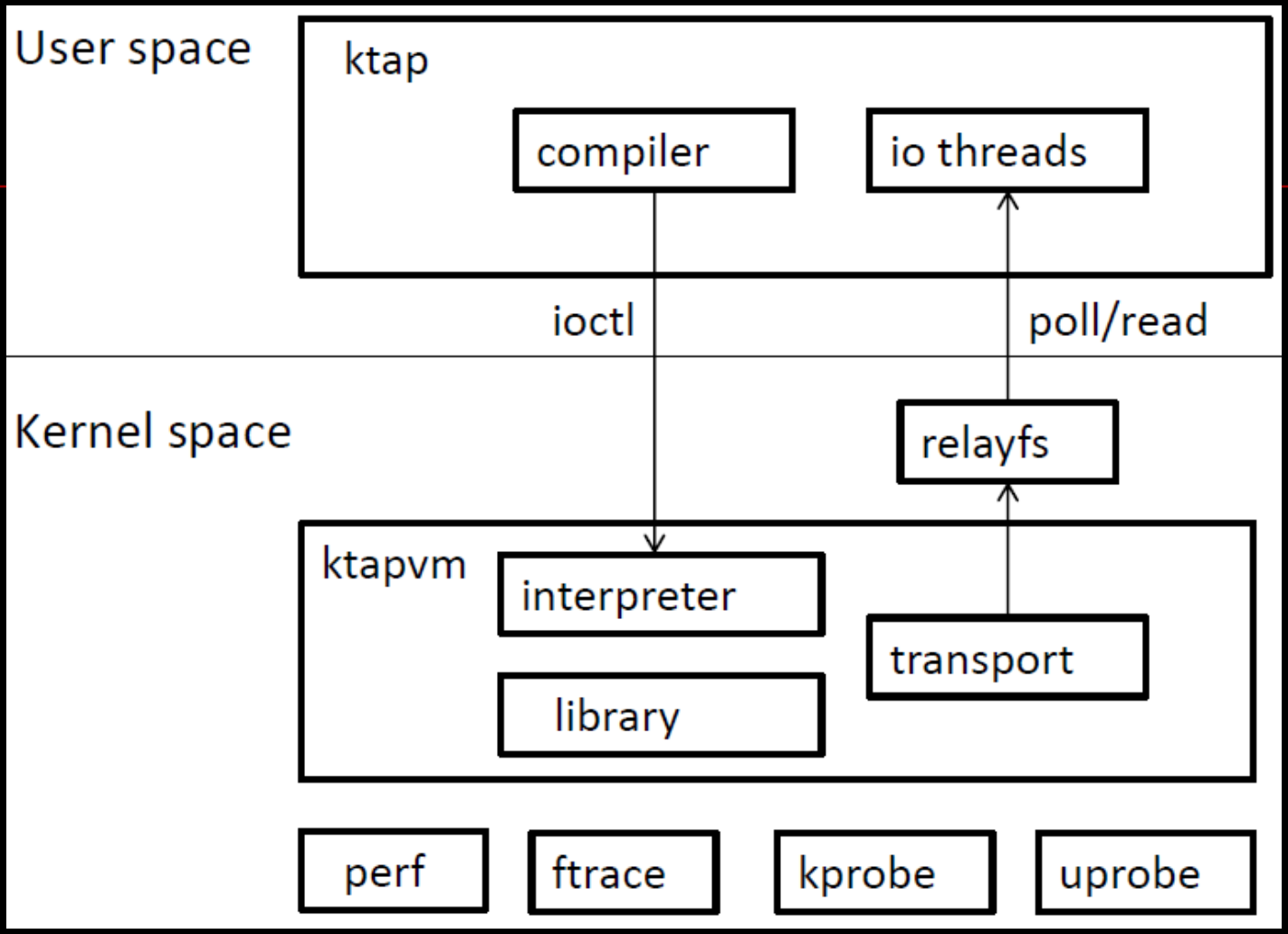
```
* ktap code is based on luajit(compiler & bytecode), so carry luajit
  copyright notices in below.

LuaJIT -- a Just-In-Time Compiler for Lua. http://luajit.org/
```

```
* Some ktap code is based on lua programming language initially,
  so carry lua own copyright notices and license terms:
  (lua's MIT license is compatible with GPL.
   ktap can redistribute as GPL v2, without violate with lua license,
   this was confirmed with official lua team)
```

History

- https://lwn.net/Articles/551314/ //Ktap — yet another kernel tracer
- https://lwn.net/Articles/572788/ //Ktap almost gets into 3.13
- https://lwn.net/Articles/595565/ //Ktap or BPF?
- https://lwn.net/Articles/595581/ //ktap and ebpf integration

Architecture



Source: "Ktap--A New Scripting Dynamic Tracing Tool For Linux", Wei Zhang(Huawei)," LinuxCon Japan 2013

V. Other In-Kernel VMs

1) KPlugs

- http://www.kplugs.org/

KPlugs is a Linux kernel module which provides an interface for dynamically executing scripts inside the Linux kernel. KPlugs uses a simple bytecode interpreter (the KPlugs Virtual Machine), and an interface that allows a user to dynamically load scripts into the kernel and execute them directly from user space. Because the interface is dynamic, it's easy to implement a user-mode library that wraps anything in the kernel!

KPlugs comes with a **Python** library that compiles a subset of the Python language to the KPlugs bytecode, and lets you easily load and execute your "kernel Python script".

The Python compiler is very basic and was made to fit our own purposes, but if it's not exactly what you are looking for, you can modify it to create your own subset.

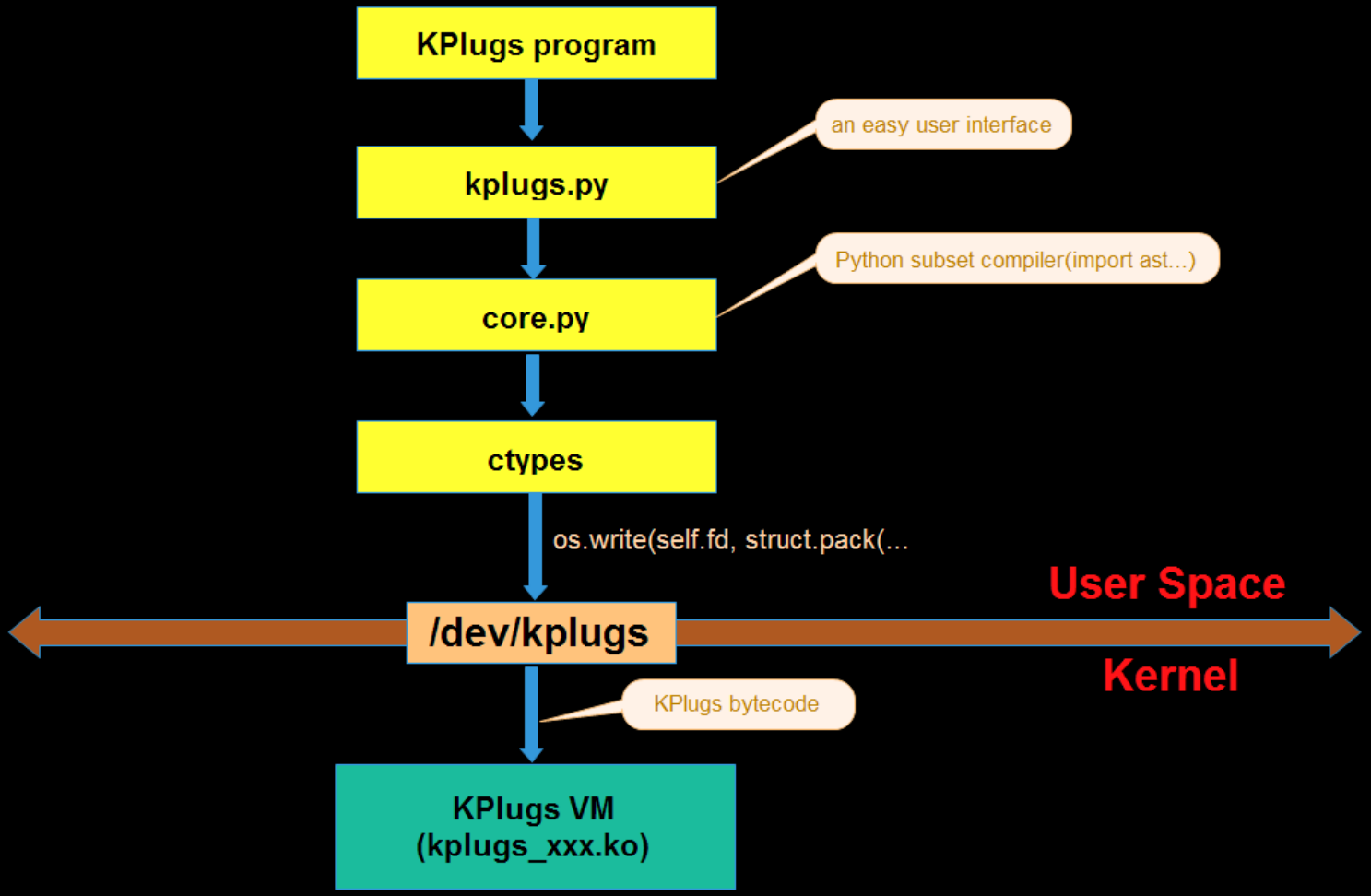
A few reasons why KPlugs is a strong and unique tool:

- The bytecode is generic and is not specific to any scripting language. If you prefer another language to Python, write a compiler for your preferred language that compiles to KPlugs bytecode, and you'll be able to run scripts written in that language in the kernel!
- When you load a function, the return value is a valid function pointer for all intents and purposes. You can call it from kernel mode or pass it as a callback to a kernel function, as long as you have the right calling convention (see the FAQ).
- The VM provides a safe environment for the bytecode to run, where every operation is checked to avoid system crashes. This works even though you are free to pass user space and kernel space buffers interchangeably to your function at your convenience; KPlugs does all the worrying for you. Of course, if you incorrectly call external functions in your bytecode, they may crash.
- The bytecode is a high-level interpreted language (supporting exceptions, for example), but its basic variable type is the standard CPU word. This allows you to write code in a high level scripting language such as Python on one hand, and interface naturally with common kernel functions on the other hand.

- https://github.com/avielw/kplugs

```
#!/usr/bin/python3

import kplugs

with kplugs.Context() as context:
    kernel_func = r'''

def hello_world(string):
    buffer(string, 0x100)
    print("%s" % string)
'''

    plug = context.Plug()
    hello_world = plug.compile(kernel_func)[0]
    hello_world("Hello World!")
```

Workflow



- How about translate Python to BPF directly?

py2bpf

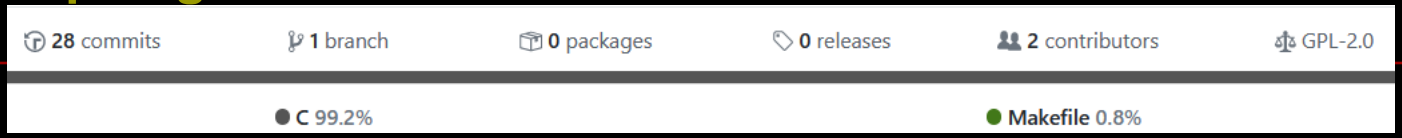
- https://github.com/facebookresearch/py2bpf
- translates functions from Python to BPF
-

```
q = py2bpf.datastructures.BpfQueue(ctypes.c_int)

@py2bpf.kprobe.probe('sys_close')
def on_sys_close(pt_regs):
    pid = py2bpf.funcs.get_current_pid_tgid() & 0xffffffff
    ptr = py2bpf.funcs.addrof(pid)
    cpuid = py2bpf.funcs.get_smp_processor_id()
    py2bpf.funcs.perf_event_output(pt_regs, q, cpuid, ptr)
    return 0


with on_sys_close():
    for pid in q:
        print('pid={}'.format(pid))
```

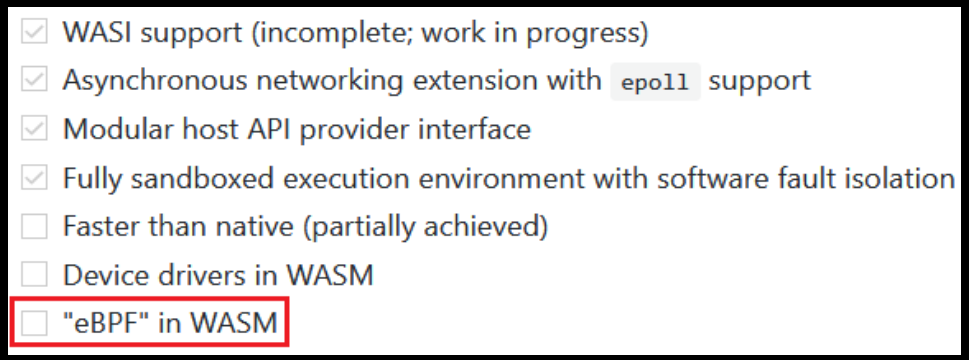
2) WASM

2.1 kernel-wasm

- https://github.com/wasmerio/kernel-wasm



- safely run WebAssembly in the Linux kernel, with faster-than-native performance
- Features



- //Running WebAssembly on the Kernel
  https://www.tuicool.com/articles/QRZzaeb

VI. New Ideas

1) LuaJIT

- http://luajit.org/
- LuaJIT has been in continuous development since 2005. It's widely considered to be **one of the fastest dynamic language implementations**. It has outperformed other dynamic languages on many cross-language benchmarks since its first release — often by a substantial margin.

  For **LuaJIT 2.0**, the whole VM has been rewritten from the ground up and relentlessly optimized for performance. It combines a **high-speed interpreter**, written in assembler, with a **state-of-the-art JIT compiler**.

  An innovative **trace compiler** is integrated with advanced, SSA-based optimizations and highly tuned code generation backends. A substantial reduction of the overhead associated with dynamic languages allows it to break into the performance range traditionally reserved for offline, static language compilers.
- https://github.com/LuaJIT
- https://github.com/openresty/luajit2

- Updated infrequently, no ARM Neon support…

Ravi

- https://the-ravi-programming-language.readthedocs.io/en/latest/
- https://github.com/dibyendumajumdar/ravi

Ravi is a derivative/dialect of Lua 5.3 with limited optional static typing and features LLVM and Eclipse OMR powered JIT compilers. The name Ravi comes from the Sanskrit word for the Sun. Interestingly a precursor to Lua was Sol which had support for static types; Sol means the Sun in Portugese.

Lua is perfect as a small embeddable dynamic language so why a derivative? Ravi extends Lua with static typing for improved performance when JIT compilation is enabled. However, the static typing is optional and therefore Lua programs are also valid Ravi programs.

There are other attempts to add static typing to Lua - e.g. Typed Lua but these efforts are mostly about adding static type checks in the language while leaving the VM unmodified. The Typed Lua effort is very similar to the approach taken by Typescript in the JavaScript world. The static typing is to aid programming in the large - the code is eventually translated to standard Lua and executed in the unmodified Lua VM.

My motivation is somewhat different - I want to enhance the VM to support more efficient operations when types are known. Type information can be exploited by JIT compilation technology to improve performance. At the same time, I want to keep the language safe and therefore usable by non-expert programmers.

Of course there is the fantastic LuaJIT implementation. Ravi has a different goal compared to LuaJIT. Ravi prioritizes ease of maintenance and support, language safety, and compatibility with Lua 5.3, over maximum performance. For more detailed comparison please refer to the documentation links below.

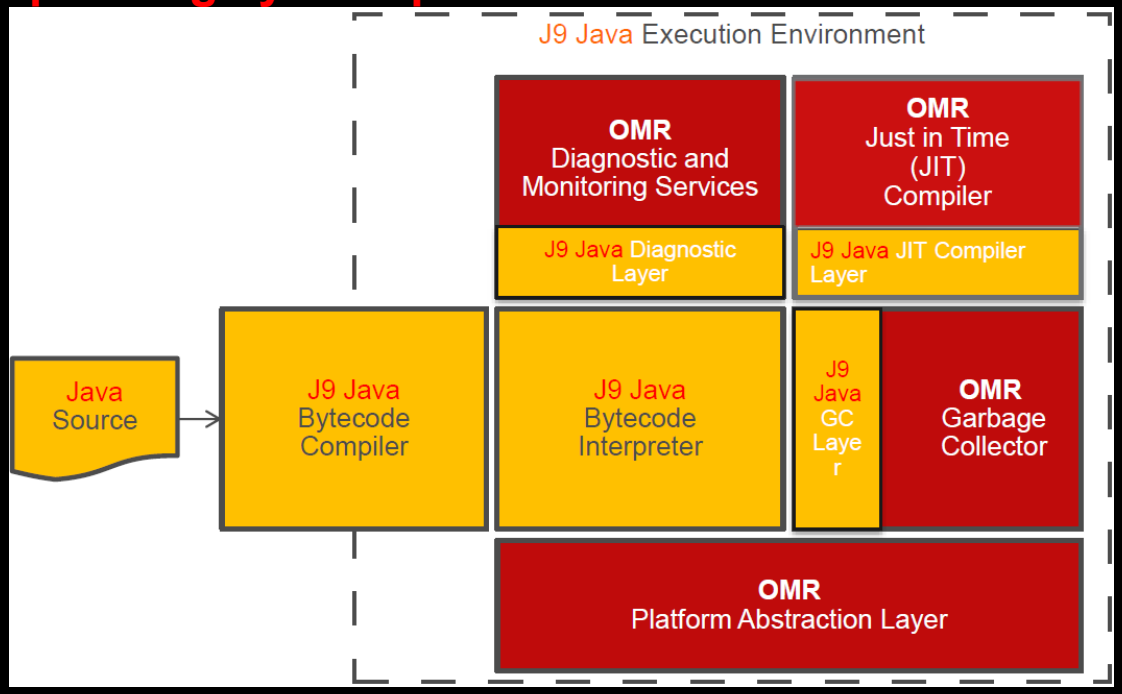
- Features

- Optional static typing - for details see the reference manual.
- Type specific bytecodes to improve performance
- Compatibility with Lua 5.3 (see Compatibility section below)
- LLVM powered JIT compiler
- Eclipse OMR powered JIT compiler
- Built-in C pre-processor, parser and JIT compiler
- A distribution with batteries.

Eclipse OMR

- http://www.eclipse.org/omr/

~~a set of open source C and C++ components that can be used to build~~ robust language runtimes that support many different hardware and operating system platforms.

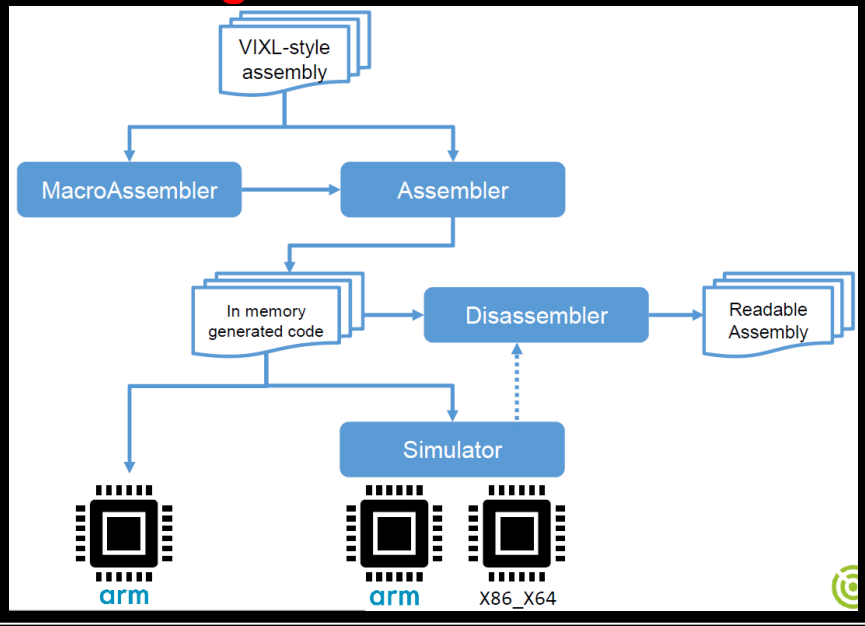


Source: "Eclipse OMR: Building Blocks for Polyglot ", Xiaoli (Shelley) Liang, TURBO 2018

- https://github.com/eclipse/openj9

VIXL

- https://git.linaro.org/arm/vixl.git
- AArch32 and AArch64 Runtime Code Generation Library
-
  - Runtime code generation library
    - Assembler
    - MacroAssembler
    - Disassembler
    - Simulator (AArch64 only)
  - Instruction coverage.
    - Core AArch32/64 v8.x
    - FP/Neon – fp16/32/64
    - SVE (on-going)
    - EL0 only
- Block Diagram



Idea

- build a new Lua runtime based on Ravi and plug the VIXL runtime assembler into target JIT compiler for ARM

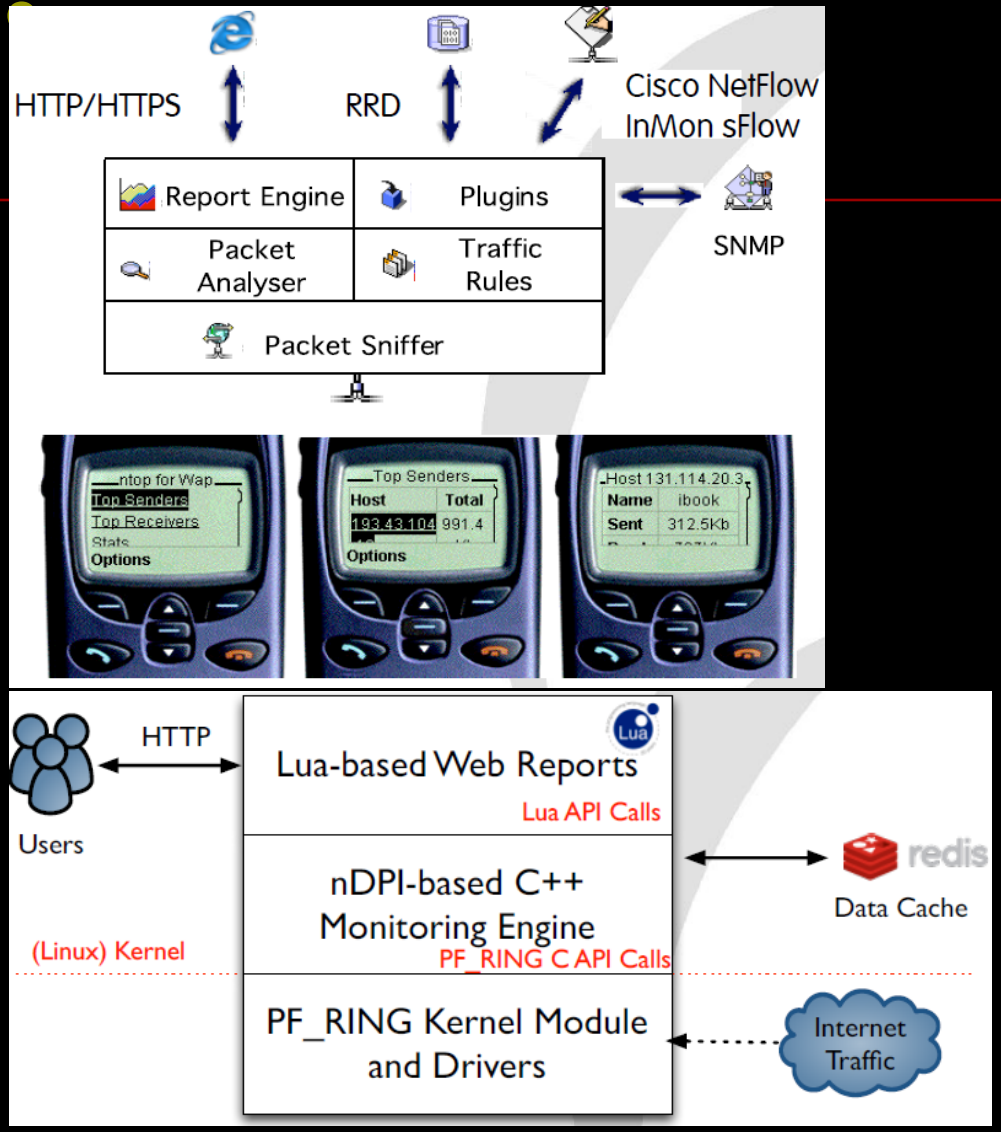
3) ntopng

- https://www.ntop.org/
- the next generation version of the original ntop, a high-speed web-based network traffic analysis and monitoring toolkit
- Features

…

- https://www.ntop.org/support/documentation/documentation/

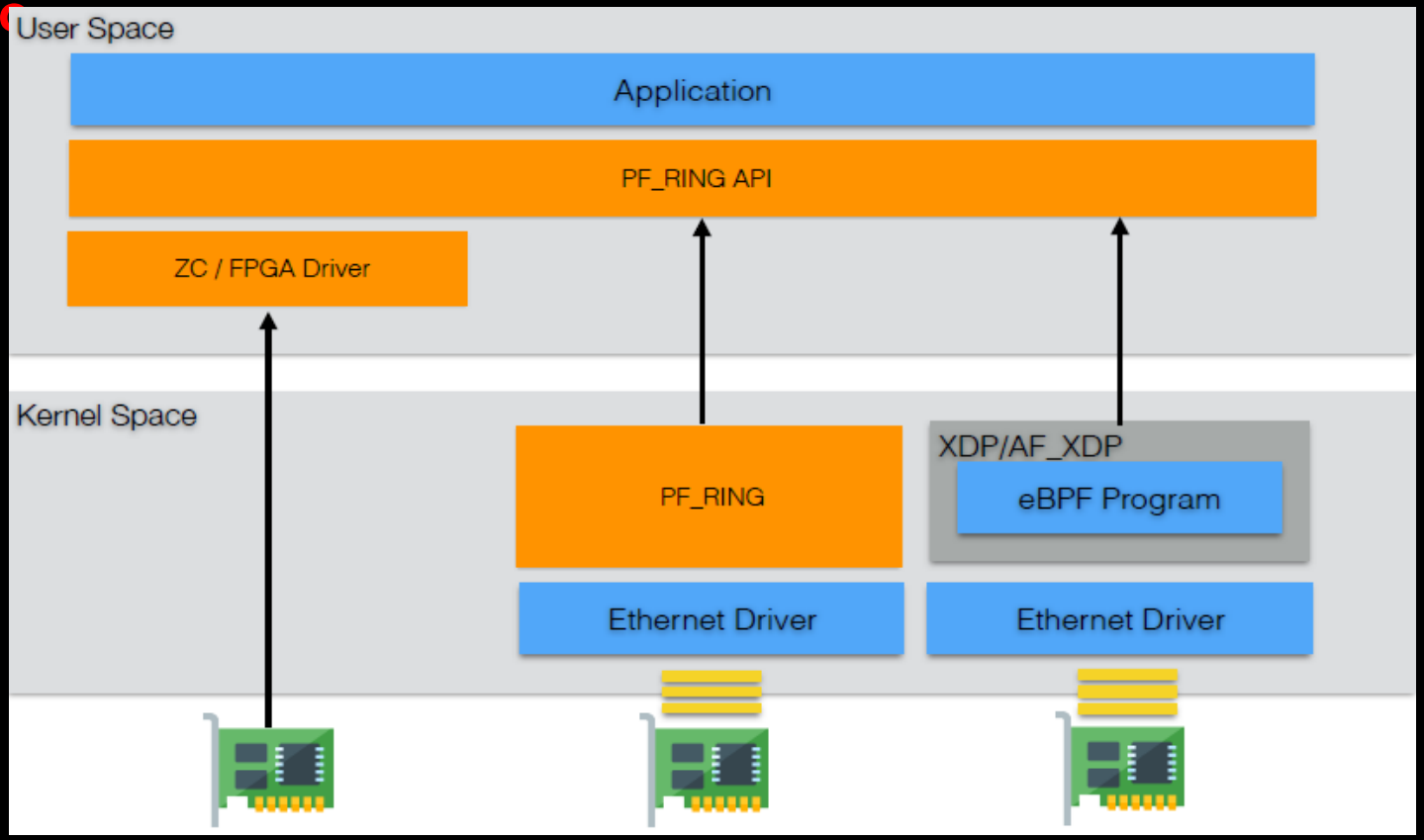
Architecture



Source: "Network Troubleshooting Using ntopng", Luca Deri, SharkFest 2015

PF_RING

- https://www.ntop.org/products/packet-capture/pf_ring
- a Linux kernel module and user-space framework that allows you to process packets at high-rates while providing a consistent API for packet processing applications
- C

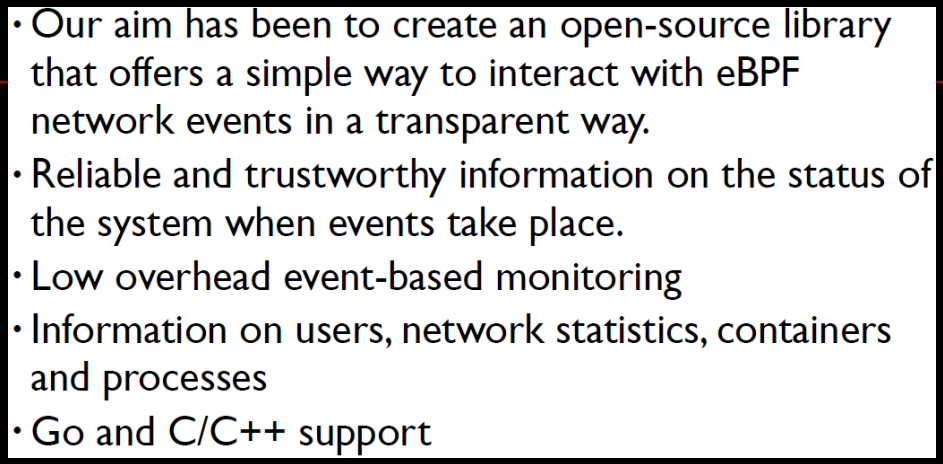


Source: "Joining Forces: PF_RING and XDP", Alfredo Cardigliano, ntopConf 2019

3.1 eBPF support

libebpfflow

- **https://github.com/ntop/libebpfflow**



Source: https://www.ntop.org/wp-content/uploads/2019/05/InfluxData_Webinar_2019.pdf

nBPF

- **https://www.ntop.org/guides/pf_ring/nbpf/nbpf.html**
  a filtering engine/SDK supporting the BPF syntax and can be used as alternative to the implementation that can be found in libpcap and inside the kernel

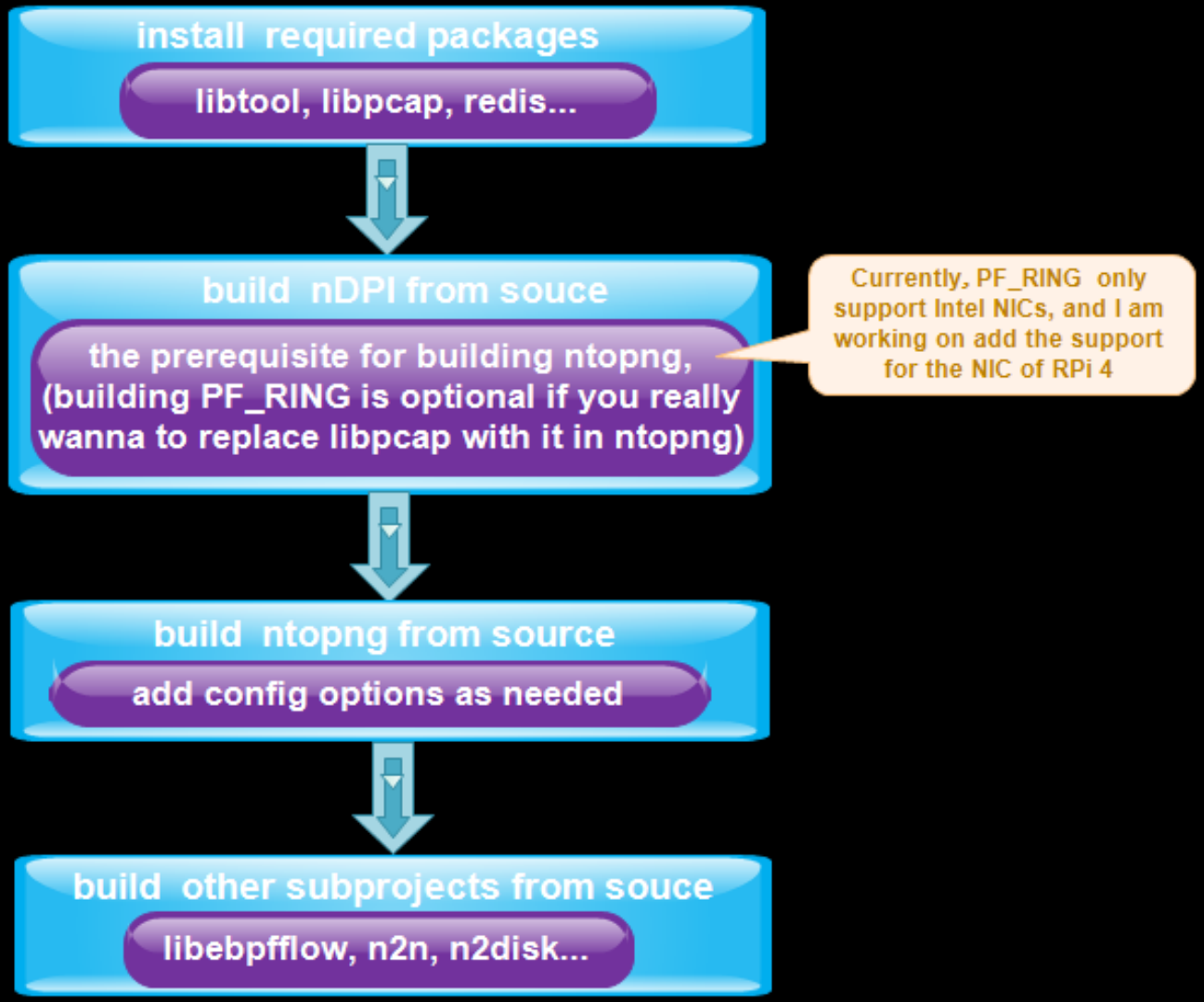
ntopng 4.x

- **ntopng 4.x will integrate system visibility with eBPF**

…

3.2 My Practice

- successfully build ntopng on RPi 4 with libpcap

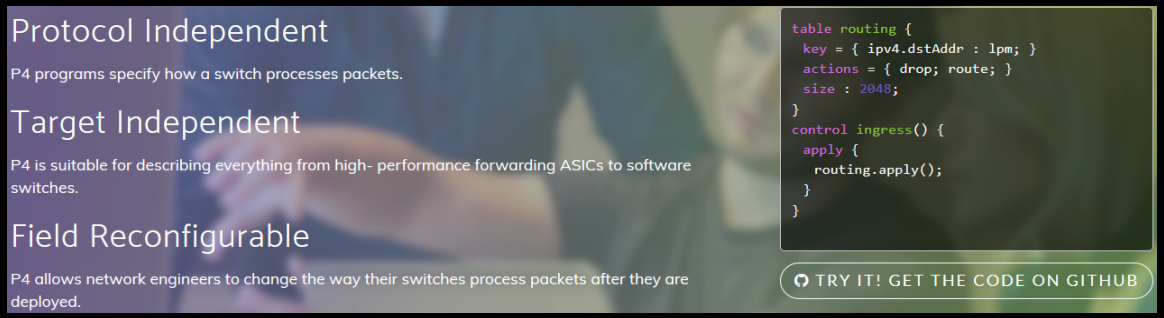


3) P4 & Stratum
3.1 P4

- https://en.wikipedia.org/wiki/P4_(programming_language)

P4 is a programming language designed to allow programming of packet forwarding planes. In contrast to a general purpose language such as C or Python, P4 is a domain-specific language with a number of constructs optimized around network data forwarding. P4 is an open-source, permissively licensed language and is maintained by a non-profit organization called the P4 Language Consortium. The language was originally described in a SIGCOMM CCR paper in 2014 titled "Programming Protocol-Independent Packet Processors"[2] – the alliterative name shortens to "P4".

- https://p4.org/



- C-like, strongly typed language
- Type and memory-safe (no pointers)
- Bounded execution (no loops)
- Statically allocated (no malloc, no recursion)
- Spec: http://github.com/p4lang/p4-spec
- Reference compiler implementation: http://github.com/p4lang/p4c (Apache 2 license)

- https://github.com/p4lang/

P4 eBPF backends

- $BCC_SRC/src/cc/frontends/p4/compiler

```
src/cc/frontends/p4/compiler
├── compilationException.py
├── ebpfAction.py
├── ebpfConditional.py
├── ebpfCounter.py
├── ebpfDeparser.py
├── ebpfInstance.py
├── ebpfParser.py
├── ebpfProgram.py
├── ebpfScalarType.py
├── ebpfStructType.py
├── ebpfTable.py
├── ebpfType.py
├── p4toEbpf.py
├── programSerializer.py
├── README.txt
├── target.py
├── topoSorting.py
└── typeFactory.py
```

```
#!/usr/bin/env python

# Copyright (c) Barefoot Networks, Inc.
# Licensed under the Apache License, Version 2.0 (the "License")

# Compiler from P4 to EBPF
# (See http://www.slideshare.net/PLUMgrid/ebpf-and-linux-networking).
# This compiler in fact generates a C source file
# which can be compiled to EBPF using the LLVM compiler
# with the ebpf target.
#
# Main entry point.
```

https://github.com/p4lang/p4-hlir/

p4-hlir only supports the P4_14 version of the P4 programming language. If you need to compile P4_16 programs, you can use the new p4lang/p4c compiler, written in C++, which supports both P4_14 and P4_16. We will keep maintaining p4-hlir for the foreseeable future, but there is no plan to add P4_16 support to it.

- $BCC_SRC/src/cc/frontends/p4

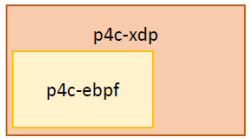
The current version of the P4 to EBPF compiler translates programs written in the version 1.1 of the P4 programming language to programs written in a restricted subset of C. The subset of C is chosen such that it should be compilable to EBPF using BCC.

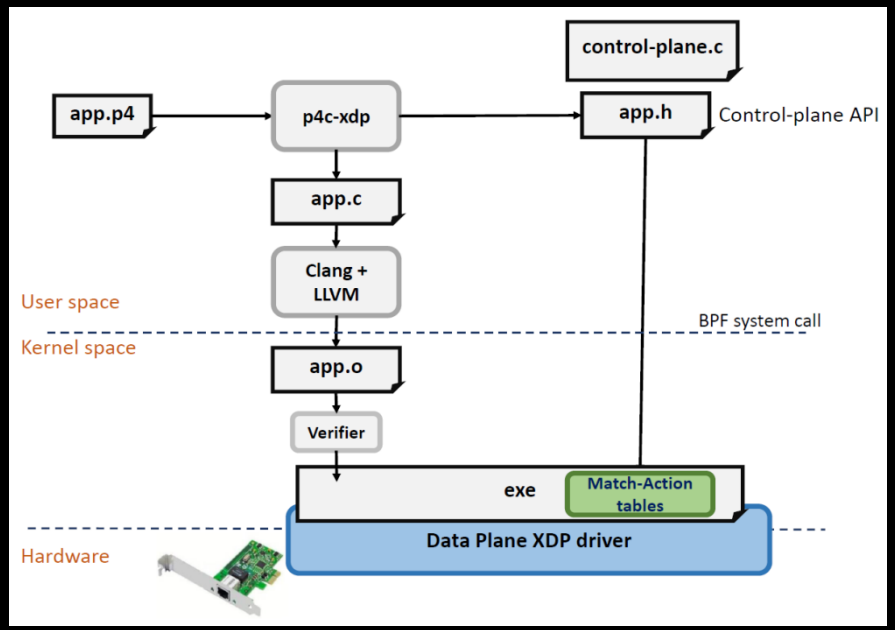
```
        -------------                -------
P4 --->  | P4-to-EBPF | ---> C ----> | BCC | --> EBPF
        -------------                -------
```

The P4 program only describes the packet processing *data plane*, that runs in the Linux kernel. The *control plane* must be separately implemented by the user. The BCC tools simplify this task considerably, by generating C and/or Python APIs that expose the dataplane/control-plane APIs.

p4c

- p4c-ebpf is part of the open-source distribution
  - http://github.com/p4lang/p4c/backends/ebpf
- p4c-xdp is a separate open-source project
  - http://github.com/vmware/p4c-xdp
  - Extension of the p4c compiler
  - Reuses much of the code
- Not production-ready
  - Needs more work
  - Known bugs and limitations
  - Generated not efficient yet

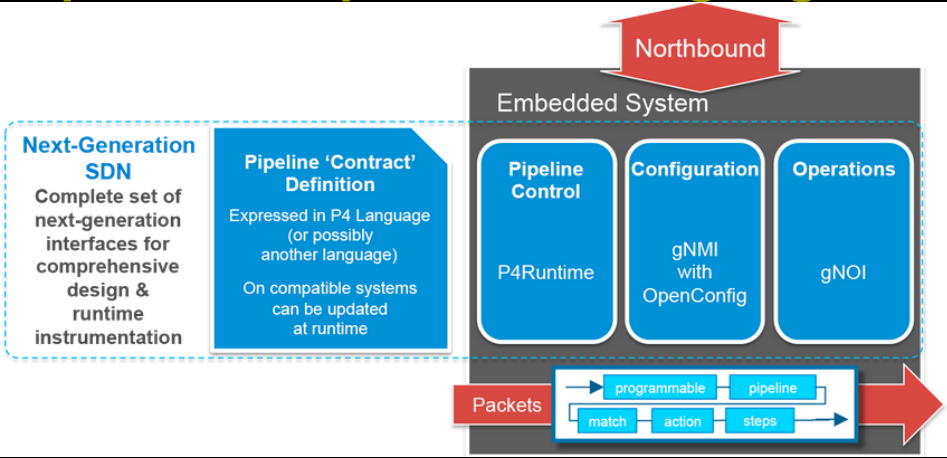




Source: "Linux Network Programming with P4", LPC 2018

3.2 Stratum

- https://www.opennetworking.org/stratum/



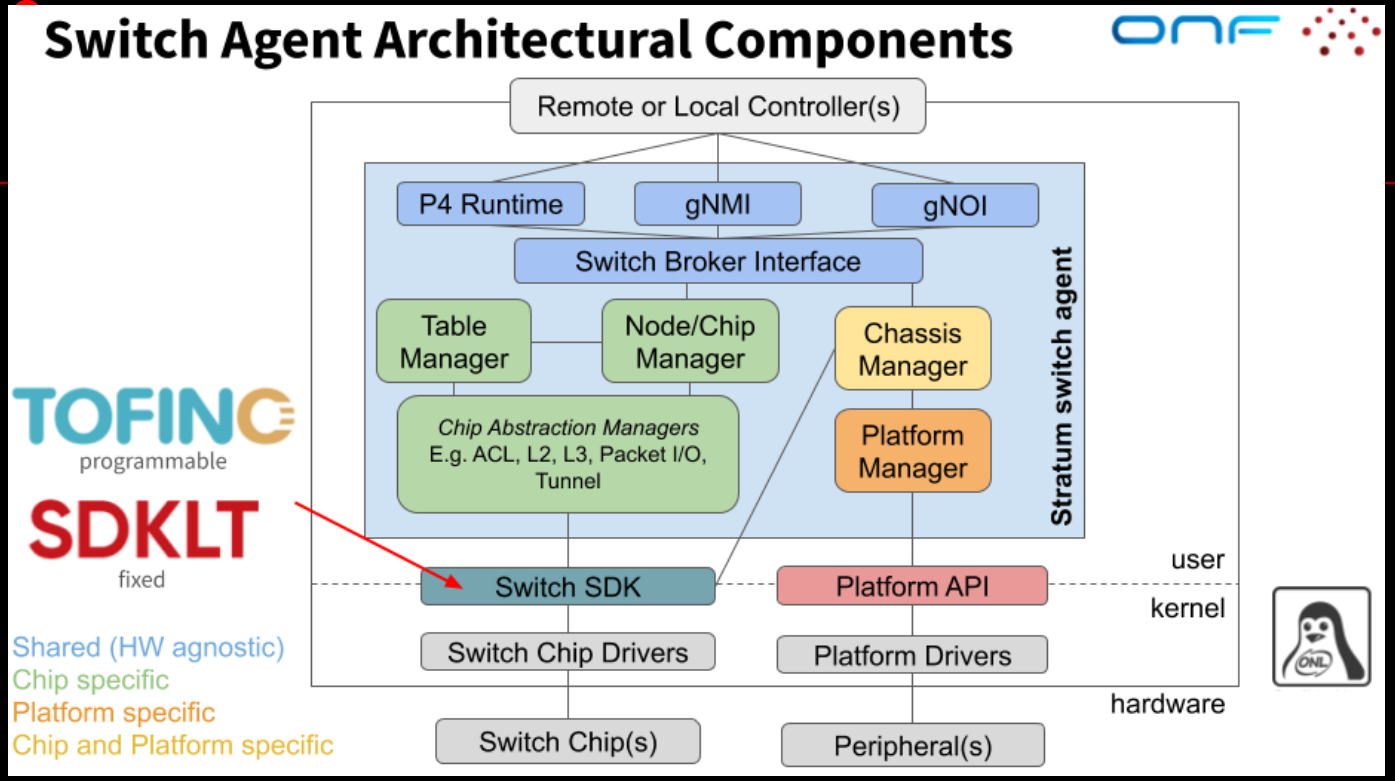
The Stratum project broadens the scope of SDN to include full lifecycle control, configuration and operations interfaces.

Envisioned as a key software component of SDN solutions of the future, Stratum implements the latest SDN-centric northbound interfaces, including P4, P4Runtime, gNMI/OpenConfig, and gNOI. It does not embed control protocols, but instead is designed to support either an external Network OS or to work with NOS functions running on the same embedded switch.

- https://github.com/stratum/stratum

Stratum is an open source silicon-independent switch operating system for software defined networks. It is building an open, minimal production-ready distribution for white box switches. Stratum exposes a set of next-generation SDN interfaces including P4Runtime and OpenConfig, enabling interchangeability of forwarding devices and programmability of forwarding behaviors. Current support includes Barefoot Tofino and Broadcom Tomahawk devices, as well as the bmv2 software switch.

Architecture



P4Runtime provides a flexible mechanism for configuring the forwarding pipeline on a network switch.

gNMI is a framework for network device management that uses gRPC as the transport mechanism.

SDKLT is used to program fixed-pipeline switches using the Tomahawk chip from Broadcom.

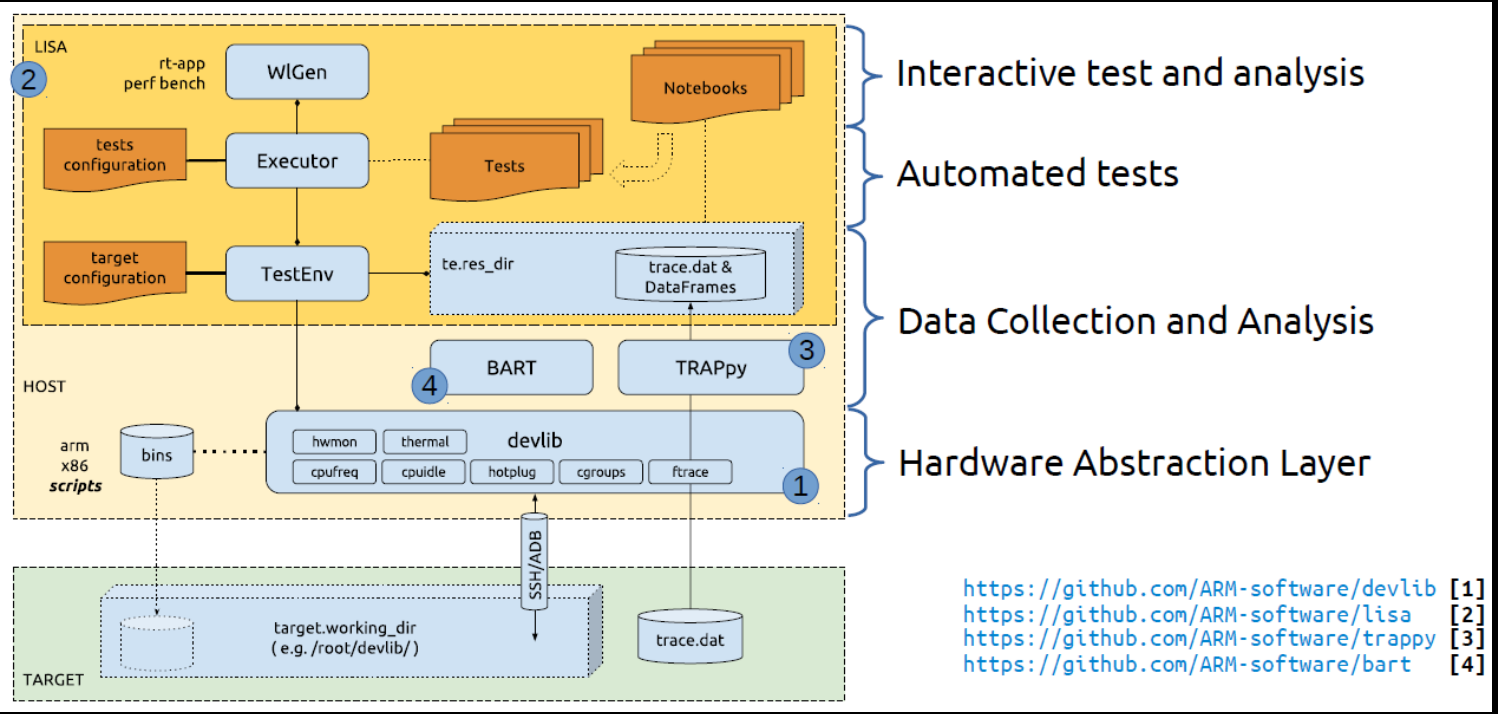
- **How about add eBPF support here?**
  P4 as a DSL on eBPF VM…

4) LISA2

- https://github.com/Yuwenfeng2019/LISA2
- extending project LISA with new designs.

Project LISA

- Linux Integrated System Analysis
- https://github.com/ARM-software/lisa



Source: “Linux Integrated System Analysis (LISA) & Friends ”, Patrick Bellasi, ELC 2016

Why is LISA2

- need a Swiss Army Knife for our development work

Project LISA(https://github.com/ARM-software/lisa) is interesting, but have some limitations from my point of view:

1. Bias towards automated testing, which has overlap in functionality with LAVA(https://www.lavasoftware.org/) from Linaro in some extent
2. Android-oriented, while it is required to adapt to a much more generalized Linux system
3. Lack of plug-in mechanism to integrated existed Kernel analysis and test projects

In LISA2, we are trying to extend LISA in the following ways which is divided into two stages in the plan: Stage I

1. In addition to Ftrace(https://en.wikipedia.org/wiki/Ftrace), combining with eBPF support, especially integrate BCC(https://github.com/iovisor/bcc)
2. Architecture redesign for better support existing Kernel analysis and test utilities, e.g. flexible plug-in mechanism and well-designed interfaces for system extension
3. Integrate the emerging Linux Kernel debugger like Drgn(https://github.com/osandov/drgn)
4. Integrate the emerging Linux Kernel test platform KernelCI(https://kernelci.org/)
5. Better support hardware debug interface like JTAG(https://en.wikipedia.org/wiki/JTAG) and SWD(https://www.arm.com/files/pdf/Serial_Wire_Debug.pdf)
6. Integrate good on-chip debugger like OpenOCD(openocd.org)

Stage II

1. More CoreSight(https://developer.arm.com/architectures/cpu-architecture/debug-visibility-and-trace/coresight-architecture) support will be added ...

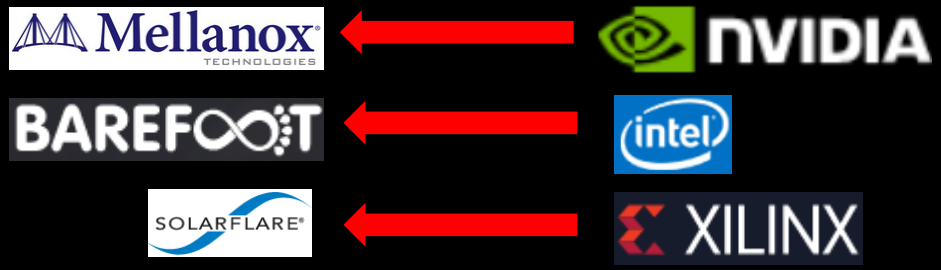
- PIs refer to my presentation "Python for Linux Kernel Debugging" at PyCon China Hangzhou (Oct 19, 2019) for details & preliminary work

VII. Wrap-up

- Hardware design and manufacturing technologies are growing rapidly in recent years, and emerging techniques like Persistent Memory(MRAM, PCRAM, ReRAM…) and SmartNIC will dramatically change the underlying system design both in hardware and software.

- User Space & Kernel Space repartition in Linux has a long history, now eBPF is driving it.
  PIs refer to my presentation "Rethinking Hyper-Converged Infrastructure for Edge Computing" at OpenInfra Days China 2019 for details & preliminary work

- The next wave of Networking Innovation is on the way



Q & A

Thanks!

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://www.kernel.org/doc/html/latest/bpf/bpf_design_QA.html
- https://www.python.org
- http://llvm.org
- http://www.brendangregg.com/ebpf
- https://en.wikipedia.org/wiki/Just-in-time_compilation
- https://www.redhat.com/en/blog/introduction-ebpf-red-hat-enterprise-linux-7
- https://developers.redhat.com/blog/2018/12/03/network-debugging-with-ebpf/
- …