CKernel(Beijing) 2017

eBPF In-kernel Virtual Machine & Cloud Computing

李枫
hkli2012@126.com
Oct 22，2017

Agenda

I. Anatomy of eBPF

- eBPF
- LLVM
- Development

II. eBPF for Kernel Instrumentation

- BCC
- Applications
- Pros & Cons

III. Cloud Computing with eBPF

- Cilium
- Load Balance
- Security
- Go-based Cloud Ecosystem

IV. eBPF on ARM

- RPi3
- IO Visor

V. Wrap-up

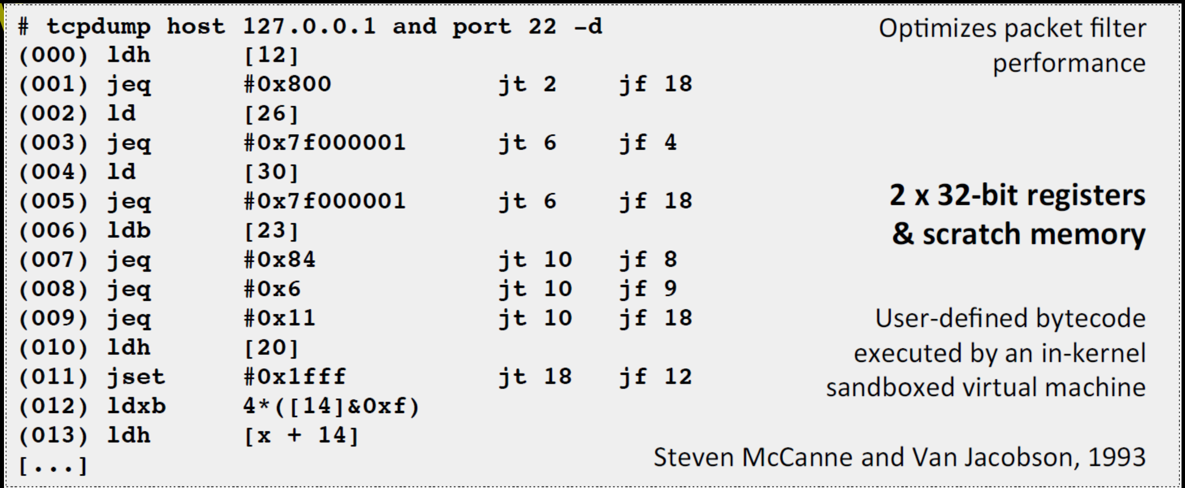
I. Anatomy of eBPF

1) eBPF

- https://en.wikipedia.org/wiki/Berkeley_Packet_Filter

BPF (Berkeley Packet Filter, aka cBPF)

- Introduced in kernel 2.1.75 (1997)
- Originally designed for packet filtering (tcpdump…)
- Apply for seccomp filters, traffic control…
- https://blog.cloudflare.com/bpf-the-forgotten-bytecode/
- V



```
# tcpdump host 127.0.0.1 and port 22 -d
(000) ldh      [12]
(001) jeq      #0x800          jt 2    jf 18
(002) ld       [26]
(003) jeq      #0x7f000001     jt 6    jf 4
(004) ld       [30]
(005) jeq      #0x7f000001     jt 6    jf 18
(006) ldb      [23]
(007) jeq      #0x84           jt 10   jf 8
(008) jeq      #0x6            jt 10   jf 9
(009) jeq      #0x11           jt 10   jf 18
(010) ldh      [20]
(011) jset     #0x1fff         jt 18   jf 12
(012) ldxb     4*([14]&0xf)
(013) ldh      [x + 14]
[...]
```

Optimizes packet filter performance

2 x 32-bit registers & scratch memory

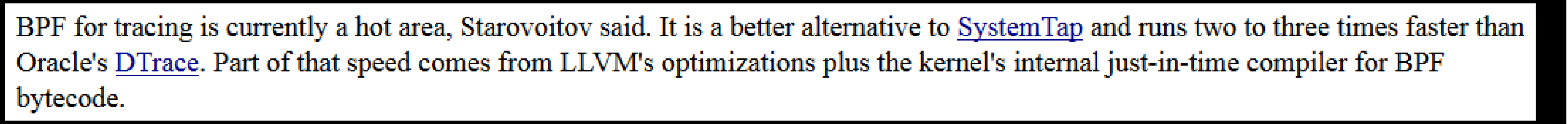
User-defined bytecode executed by an in-kernel sandboxed virtual machine

Steven McCanne and Van Jacobson, 1993

Source: https://www.slideshare.net/brendangregg/kernel-recipes-2017-performance-analysis-with-bpf

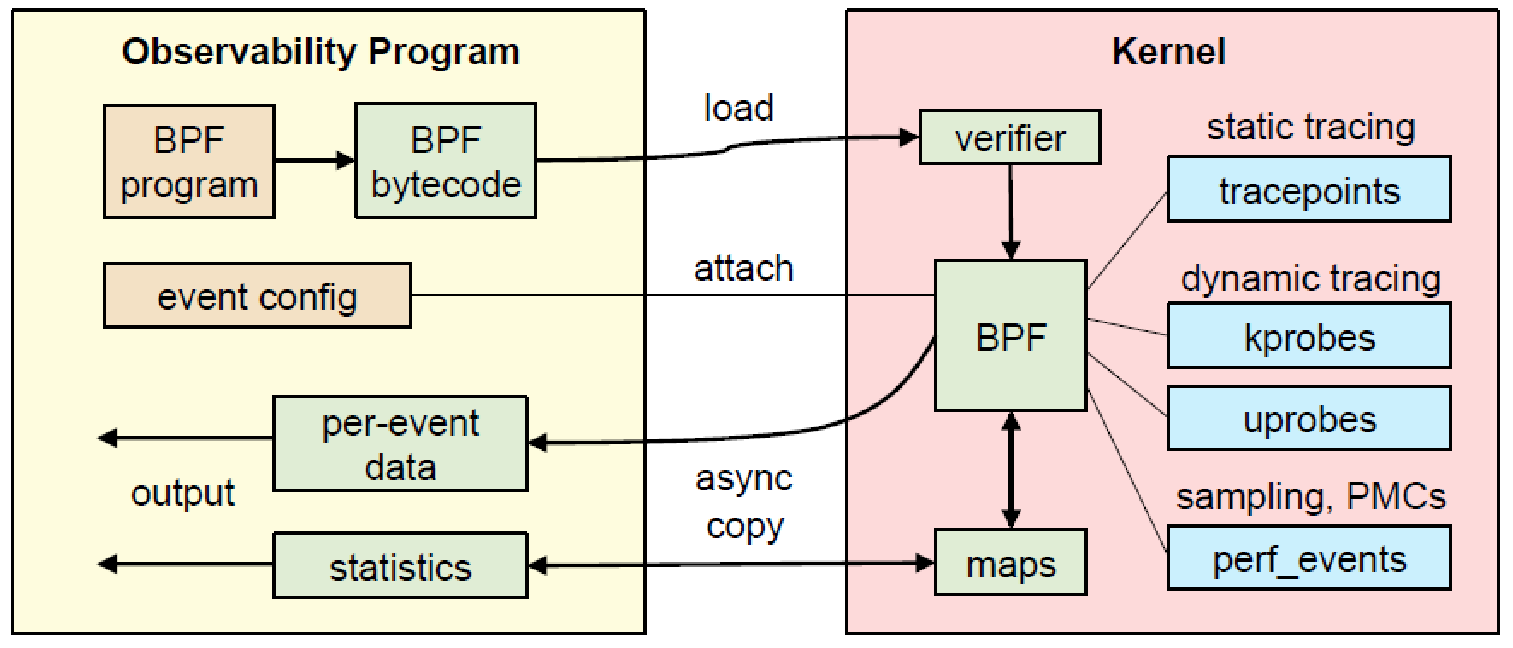
eBPF (extended BPF)

- 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
- https://lwn.net/Articles/655544



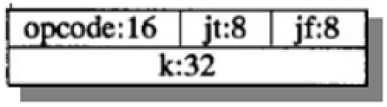
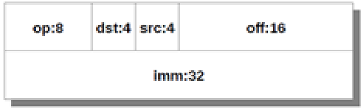
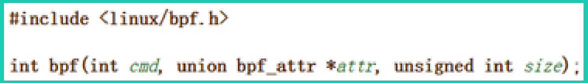
BPF for tracing is currently a hot area, Starovoitov said. It is a better alternative to SystemTap and runs two to three times faster than Oracle's DTrace. Part of that speed comes from LLVM's optimizations plus the kernel's internal just-in-time compiler for BPF bytecode.

- it changes the old ways for Kernel instrumentation



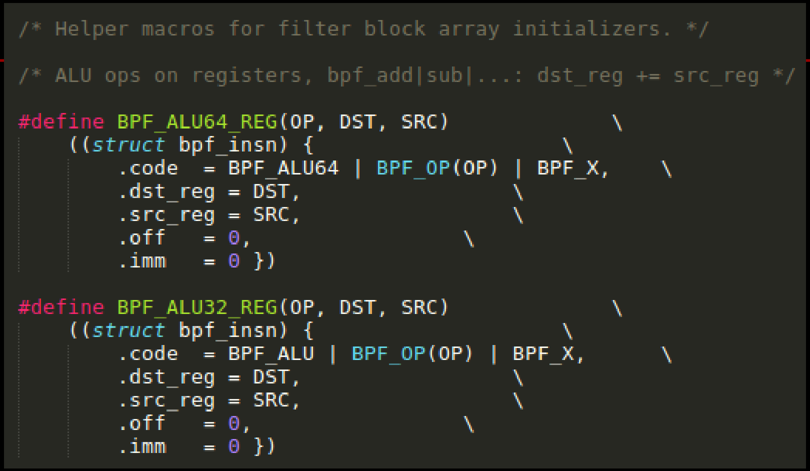
Source: https://www.slideshare.net/brendangregg/kernel-recipes-2017-performance-analysis-with-bpf

Comparison

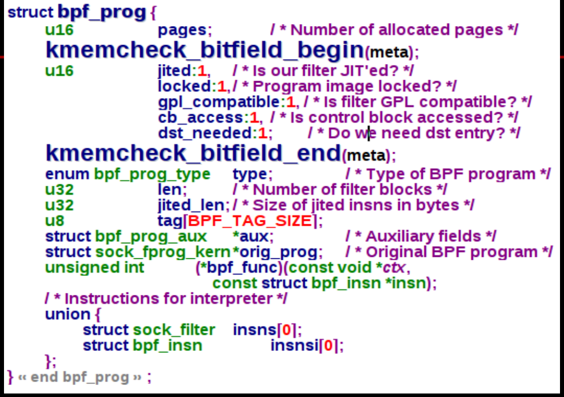
	cBPF	eBPF
Register	**Two 32 bit registers:** A: accumulator X: indexing	**Eleven 64 bit registers:** R0: return value/exit value R1-R5: arguments R6-R9: callee saved registers R10: read-only frame pointer
Instruction	**~30**  opcode:16 | jt:8 | jf:8 k:32	**~90**  op:8 | dst:4 | src:4 | off:16 imm:32
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...)**

Internal

- $KERNEL_SRC/Documentation/networking/filter.txt
- $KERNEL_SRC/include/linux/filter.h

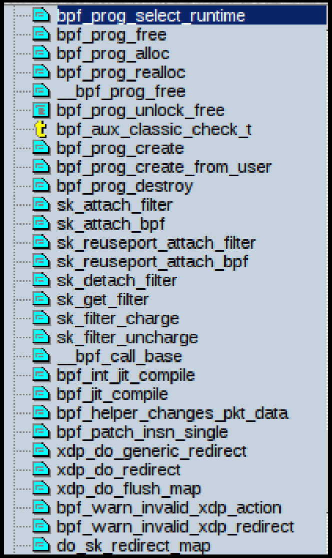


```
/* Helper macros for filter block array initializers. */

/* ALU ops on registers, bpf_add|sub|...: dst_reg += src_reg */

#define BPF_ALU64_REG(OP, DST, SRC)                     \
    ((struct bpf_insn) {                                \
        .code  = BPF_ALU64 | BPF_OP(OP) | BPF_X,        \
        .dst_reg = DST,                                 \
        .src_reg = SRC,                                 \
        .off   = 0,                                     \
        .imm   = 0 })

#define BPF_ALU32_REG(OP, DST, SRC)                     \
    ((struct bpf_insn) {                                \
        .code  = BPF_ALU | BPF_OP(OP) | BPF_X,          \
        .dst_reg = DST,                                 \
        .src_reg = SRC,                                 \
        .off   = 0,                                     \
        .imm   = 0 })
```



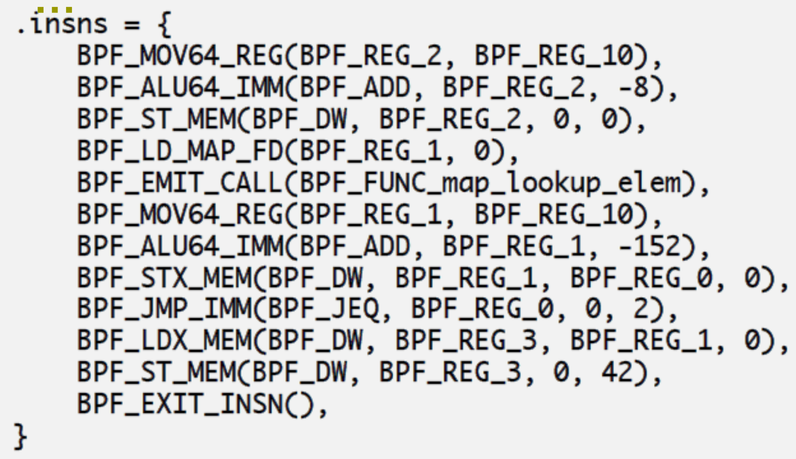
```
struct bpf_prog {
    u16                 pages;          /* Number of allocated pages */
    kmemcheck_bitfield_begin(meta);
    u16                 jited:1,    /* Is our filter JIT'ed? */
                        locked:1,/* Program image locked? */
                        gpl_compatible:1, /* Is filter GPL compatible? */
                        cb_access:1, /* Is control block accessed? */
                        dst_needed:1;      /* Do we need dst entry? */
    kmemcheck_bitfield_end(meta);
    enum bpf_prog_type    type;             /* Type of BPF program */
    u32                 len;        /* Number of filter blocks */
    u32                 jited_len;/* Size of jited insns in bytes */
    u8                  tag[BPF_TAG_SIZE];
    struct bpf_prog_aux   *aux;             /* Auxiliary fields */
    struct sock_fprog_kern*orig_prog;     /* Original BPF program */
    unsigned int          (*bpf_func)(const void *ctx,
                              const struct bpf_insn *insn);
    /* Instructions for interpreter */
    union {
        struct sock_filter   insns[0];
        struct bpf_insn          insnsi[0];
    };
} « end bpf_prog » ;
```

...



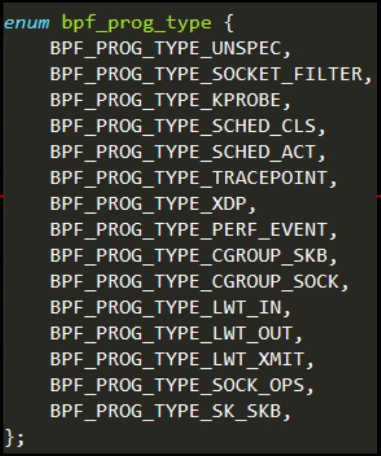
```
bpf_prog_select_runtime
bpf_prog_free
bpf_prog_alloc
bpf_prog_realloc
__bpf_prog_free
bpf_prog_unlock_free
bpf_aux_classic_check_t
bpf_prog_create
bpf_prog_create_from_user
bpf_prog_destroy
sk_attach_filter
sk_attach_bpf
sk_reuseport_attach_filter
sk_reuseport_attach_bpf
sk_detach_filter
sk_get_filter
sk_filter_charge
sk_filter_uncharge
__bpf_call_base
bpf_int_jit_compile
bpf_jit_compile
bpf_helper_changes_pkt_data
bpf_patch_insn_single
xdp_do_generic_redirect
xdp_do_redirect
xdp_do_flush_map
bpf_warn_invalid_xdp_action
bpf_warn_invalid_xdp_redirect
do_sk_redirect_map
```

...

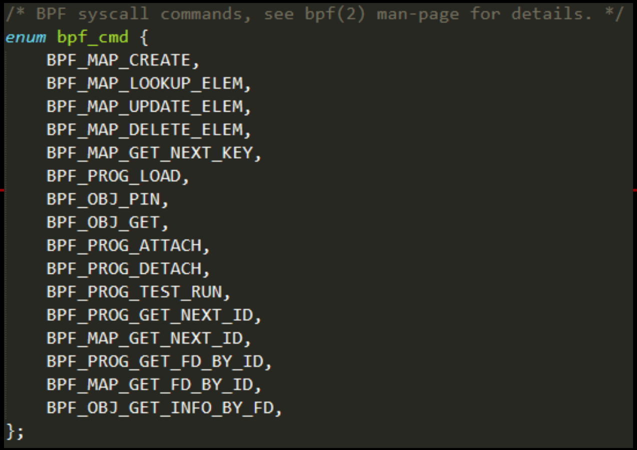


```
...
.insns = {
    BPF_MOV64_REG(BPF_REG_2, BPF_REG_10),
    BPF_ALU64_IMM(BPF_ADD, BPF_REG_2, -8),
    BPF_ST_MEM(BPF_DW, BPF_REG_2, 0, 0),
    BPF_LD_MAP_FD(BPF_REG_1, 0),
    BPF_EMIT_CALL(BPF_FUNC_map_lookup_elem),
    BPF_MOV64_REG(BPF_REG_1, BPF_REG_10),
    BPF_ALU64_IMM(BPF_ADD, BPF_REG_1, -152),
    BPF_STX_MEM(BPF_DW, BPF_REG_1, BPF_REG_0, 0),
    BPF_JMP_IMM(BPF_JEQ, BPF_REG_0, 0, 2),
    BPF_LDX_MEM(BPF_DW, BPF_REG_3, BPF_REG_1, 0),
    BPF_ST_MEM(BPF_DW, BPF_REG_3, 0, 42),
    BPF_EXIT_INSN(),
}
```

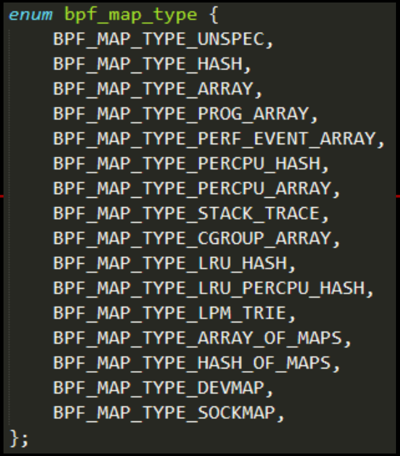
- $KERNEL_SRC/include/uapi/linux/bpf.h



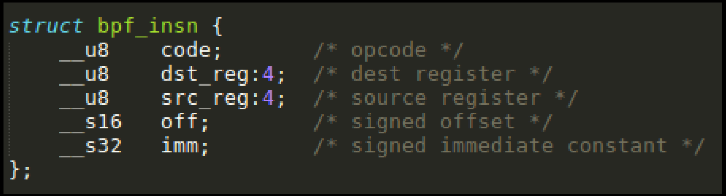
```
enum bpf_prog_type {
    BPF_PROG_TYPE_UNSPEC,
    BPF_PROG_TYPE_SOCKET_FILTER,
    BPF_PROG_TYPE_KPROBE,
    BPF_PROG_TYPE_SCHED_CLS,
    BPF_PROG_TYPE_SCHED_ACT,
    BPF_PROG_TYPE_TRACEPOINT,
    BPF_PROG_TYPE_XDP,
    BPF_PROG_TYPE_PERF_EVENT,
    BPF_PROG_TYPE_CGROUP_SKB,
    BPF_PROG_TYPE_CGROUP_SOCK,
    BPF_PROG_TYPE_LWT_IN,
    BPF_PROG_TYPE_LWT_OUT,
    BPF_PROG_TYPE_LWT_XMIT,
    BPF_PROG_TYPE_SOCK_OPS,
    BPF_PROG_TYPE_SK_SKB,
};
```



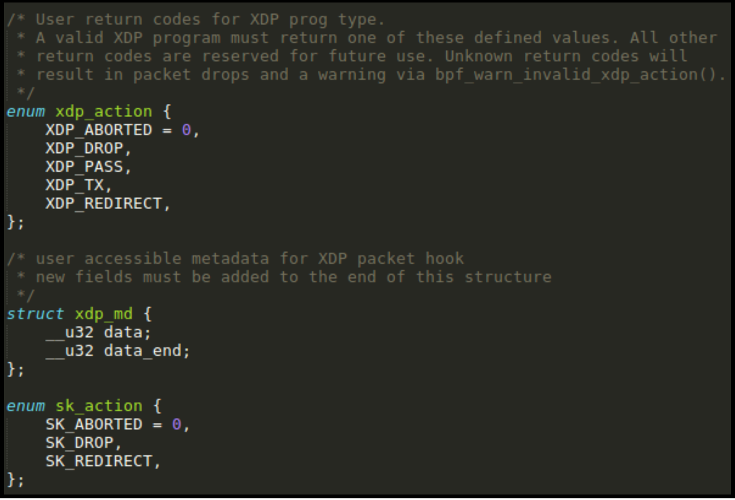
```
/* BPF syscall commands, see bpf(2) man-page for details. */
enum bpf_cmd {
    BPF_MAP_CREATE,
    BPF_MAP_LOOKUP_ELEM,
    BPF_MAP_UPDATE_ELEM,
    BPF_MAP_DELETE_ELEM,
    BPF_MAP_GET_NEXT_KEY,
    BPF_PROG_LOAD,
    BPF_OBJ_PIN,
    BPF_OBJ_GET,
    BPF_PROG_ATTACH,
    BPF_PROG_DETACH,
    BPF_PROG_TEST_RUN,
    BPF_PROG_GET_NEXT_ID,
    BPF_MAP_GET_NEXT_ID,
    BPF_PROG_GET_FD_BY_ID,
    BPF_MAP_GET_FD_BY_ID,
    BPF_OBJ_GET_INFO_BY_FD,
};
```



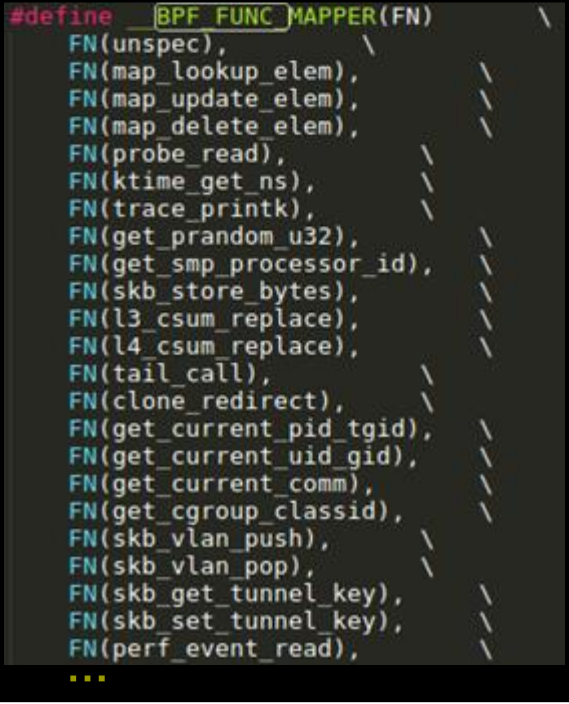
```
enum bpf_map_type {
    BPF_MAP_TYPE_UNSPEC,
    BPF_MAP_TYPE_HASH,
    BPF_MAP_TYPE_ARRAY,
    BPF_MAP_TYPE_PROG_ARRAY,
    BPF_MAP_TYPE_PERF_EVENT_ARRAY,
    BPF_MAP_TYPE_PERCPU_HASH,
    BPF_MAP_TYPE_PERCPU_ARRAY,
    BPF_MAP_TYPE_STACK_TRACE,
    BPF_MAP_TYPE_CGROUP_ARRAY,
    BPF_MAP_TYPE_LRU_HASH,
    BPF_MAP_TYPE_LRU_PERCPU_HASH,
    BPF_MAP_TYPE_LPM_TRIE,
    BPF_MAP_TYPE_ARRAY_OF_MAPS,
    BPF_MAP_TYPE_HASH_OF_MAPS,
    BPF_MAP_TYPE_DEVMAP,
    BPF_MAP_TYPE_SOCKMAP,
};
```



```
struct bpf_insn {
    __u8    code;       /* opcode */
    __u8    dst_reg:4;  /* dest register */
    __u8    src_reg:4;  /* source register */
    __s16   off;        /* signed offset */
    __s32   imm;        /* signed immediate constant */
};
```



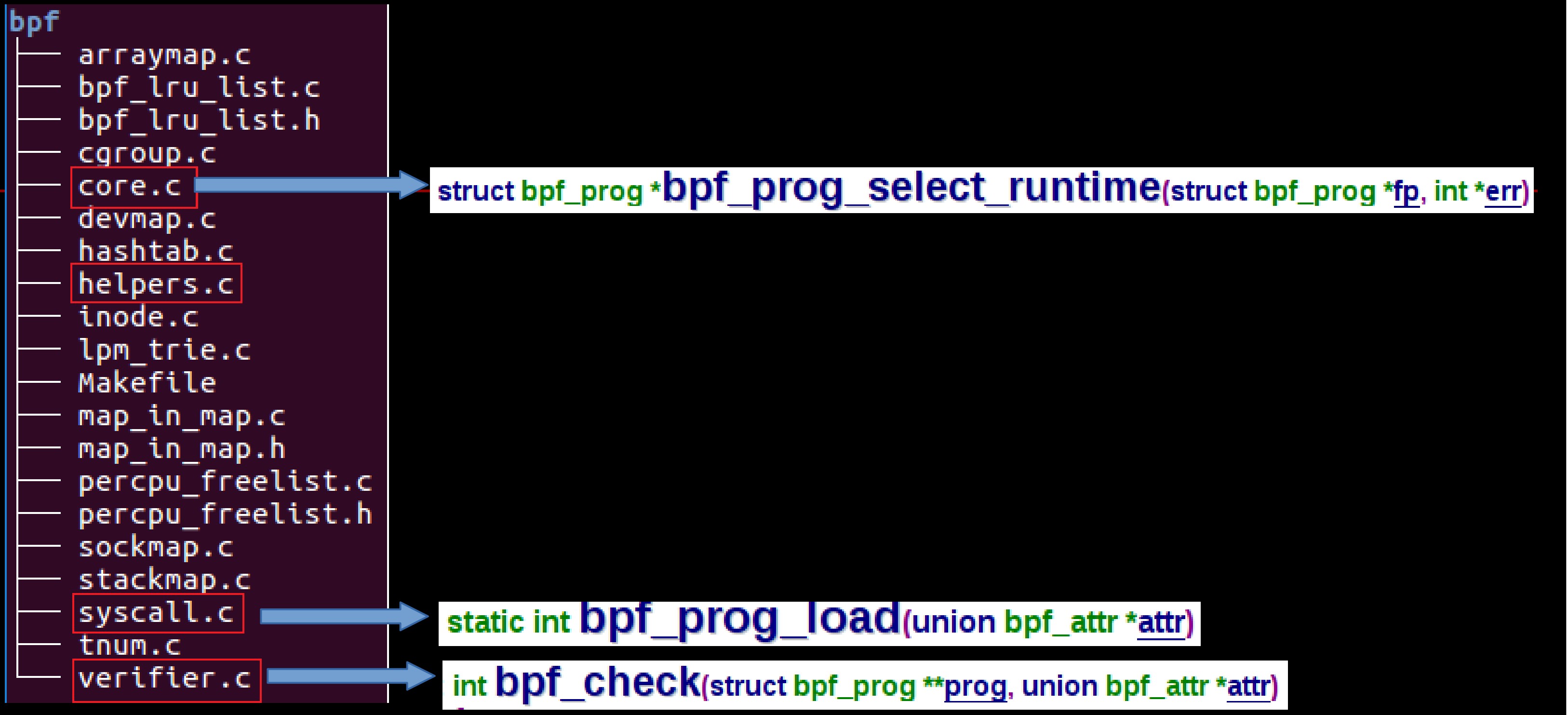
```
/* User return codes for XDP prog type.
 * A valid XDP program must return one of these defined values. All other
 * return codes are reserved for future use. Unknown return codes will
 * result in packet drops and a warning via bpf_warn_invalid_xdp_action().
 */
enum xdp_action {
    XDP_ABORTED = 0,
    XDP_DROP,
    XDP_PASS,
    XDP_TX,
    XDP_REDIRECT,
};

/* user accessible metadata for XDP packet hook
 * new fields must be added to the end of this structure
 */
struct xdp_md {
    __u32 data;
    __u32 data_end;
};

enum sk_action {
    SK_ABORTED = 0,
    SK_DROP,
    SK_REDIRECT,
};
```



```
#define __BPF_FUNC_MAPPER(FN)           \
    FN(unspec),                 \
    FN(map_lookup_elem),            \
    FN(map_update_elem),            \
    FN(map_delete_elem),            \
    FN(probe_read),             \
    FN(ktime_get_ns),           \
    FN(trace_printk),           \
    FN(get_prandom_u32),            \
    FN(get_smp_processor_id),       \
    FN(skb_store_bytes),            \
    FN(l3_csum_replace),            \
    FN(l4_csum_replace),            \
    FN(tail_call),              \
    FN(clone_redirect),         \
    FN(get_current_pid_tgid),       \
    FN(get_current_uid_gid),        \
    FN(get_current_comm),           \
    FN(get_cgroup_classid),         \
    FN(skb_vlan_push),          \
    FN(skb_vlan_pop),           \
    FN(skb_get_tunnel_key),         \
    FN(skb_set_tunnel_key),         \
    FN(perf_event_read),            \
```

...

- $KERNEL_SRC/kernel/bpf



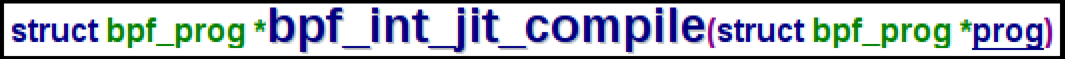
```
bpf
├── arraymap.c
├── bpf_lru_list.c
├── bpf_lru_list.h
├── cgroup.c
├── core.c
├── devmap.c
├── hashtab.c
├── helpers.c
├── inode.c
├── lpm_trie.c
├── Makefile
├── map_in_map.c
├── map_in_map.h
├── percpu_freelist.c
├── percpu_freelist.h
├── sockmap.c
├── stackmap.c
├── syscall.c
├── tnum.c
└── verifier.c
```

```
struct bpf_prog *bpf_prog_select_runtime(struct bpf_prog *fp, int *err)
```

```
static int bpf_prog_load(union bpf_attr *attr)
```

```
int bpf_check(struct bpf_prog **prog, union bpf_attr *attr)
```

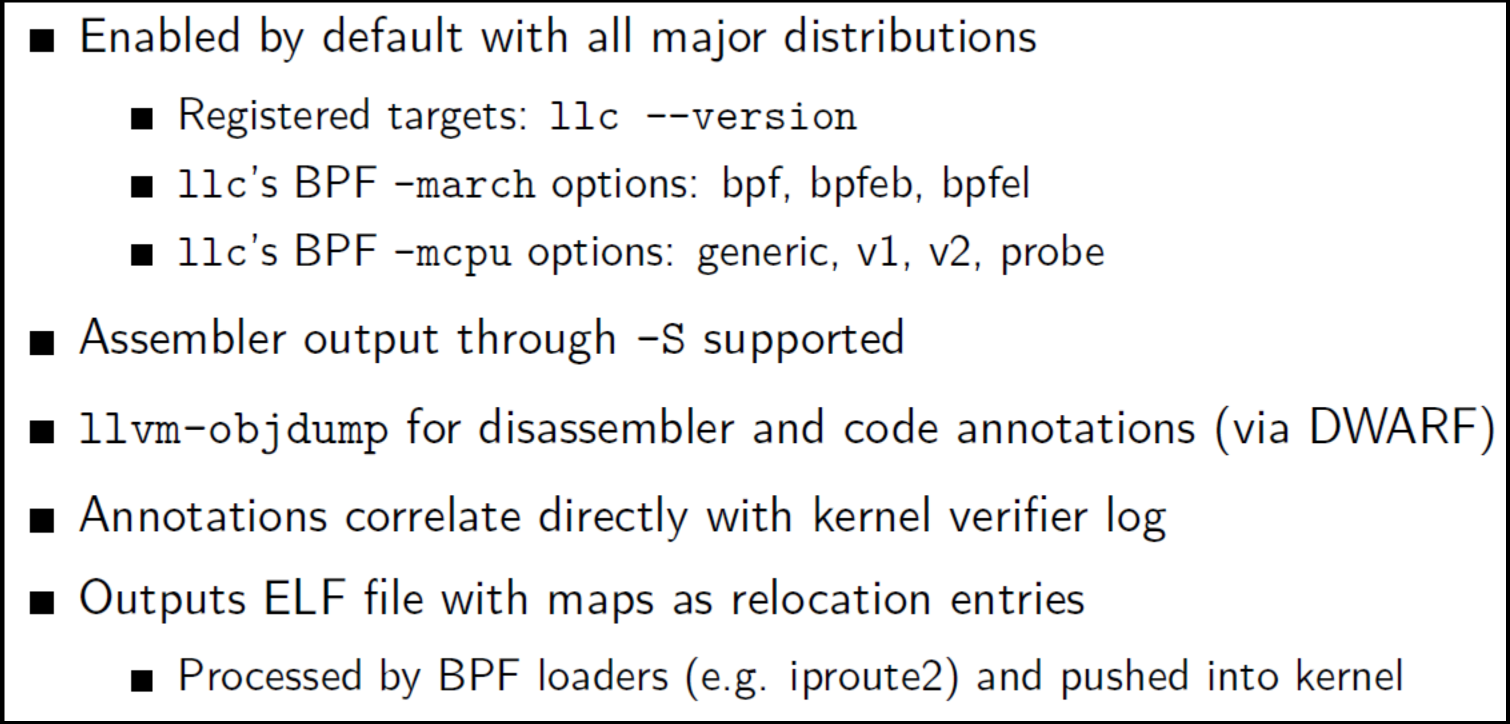
- $KERNEL_SRC/arch/$ARCH/net/bpf_jit_comp.c
  $KERNEL_SRC/arch/$ARCH/net/ebpf_jit.c
  …



```
struct bpf_prog *bpf_int_jit_compile(struct bpf_prog *prog)
```

2) LLVM

- eBPF backend firstly introduced in LLVM 3.7 release
- https://reviews.llvm.org/D6494
- http://llvm.org/docs/CodeGenerator.html#the-extended-berkeley-packet-filter-ebpf-backend
- $LLVM_SRC/lib/Target/BPF



- 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

- https://en.wikipedia.org/wiki/LLVM
- http://clang.llvm.org/

GPL v3	UIUC, MIT
Front-end: CC1 / CPP	Front-end: Clang
ld.bfd / ld.gold	lld / mclinker
gdb	lldb
as / objdump	MC layer
libstdc++	libc++
libsupc++	libc++abi
libgcc	libcompiler-rt
libgccjit	libLLVMMCJIT

How is LLVM being used today?
XCode, Swift
FreeBSD, OpenMandriva Lx
Android
Debian experimenting with Clang as an additional compiler
…

Clang Goals

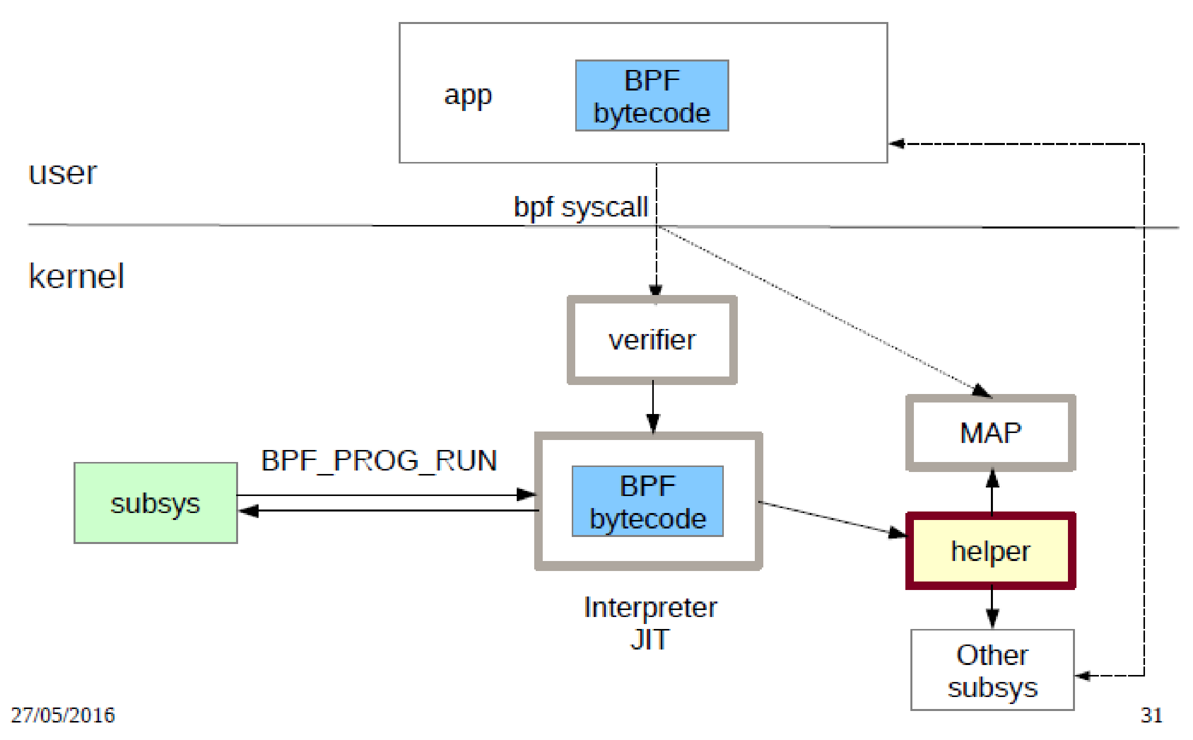
- GCC compatibility
- Fast compilation and low memory footprints
- Can reduce the linking time
- User friendly diagnostics
- Tooling
  - – static analyzers
  - – sanitizers

$KERNEL_SRC/samples/bpf/Makefile

3) Development

- Methods
  1) eBPF assembly
  2) BCC

  …
- BPF Programming Flow

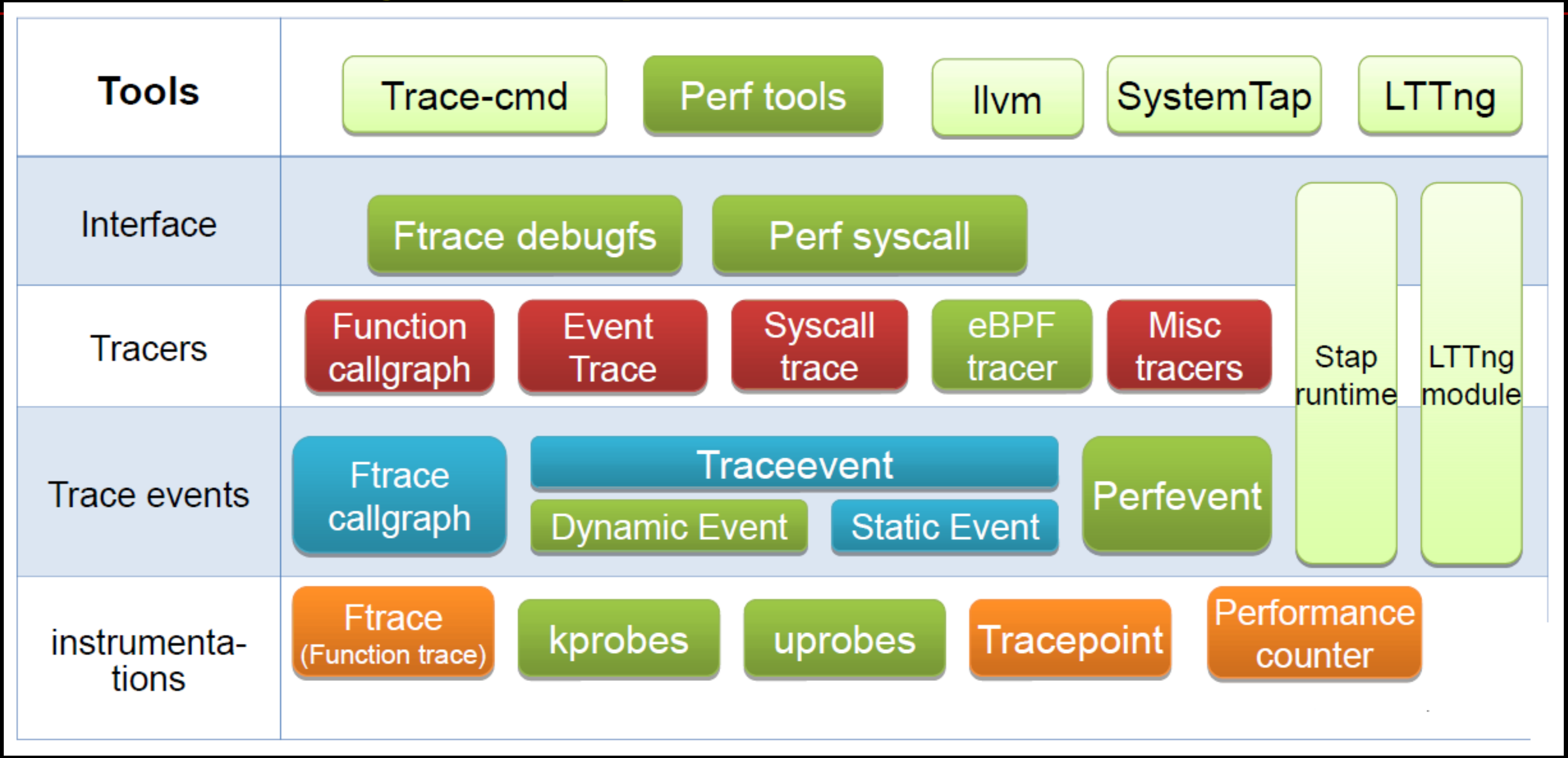


Source: http://www.slideshare.net/vh21/meet-cutebetweenebpfandtracing

2. eBPF for Kernel Instrumentation

Overview

- The Tracing Landscape



Tools	Trace-cmd	Perf tools	llvm	SystemTap	LTTng		
Interface	Ftrace debugfs	Perf syscall				Stap runtime	LTTng module
Tracers	Function callgraph	Event Trace	Syscall trace	eBPF tracer	Misc tracers		
Trace events	Ftrace callgraph	Traceevent Dynamic Event Static Event	Perfevent				
instrumenta-tions	Ftrace (Function trace)	kprobes	uprobes	Tracepoint	Performance counter		

Source: http://tracingsummit.org/w/images/8/8c/TracingSummit2015-DynamicProbes.pdf

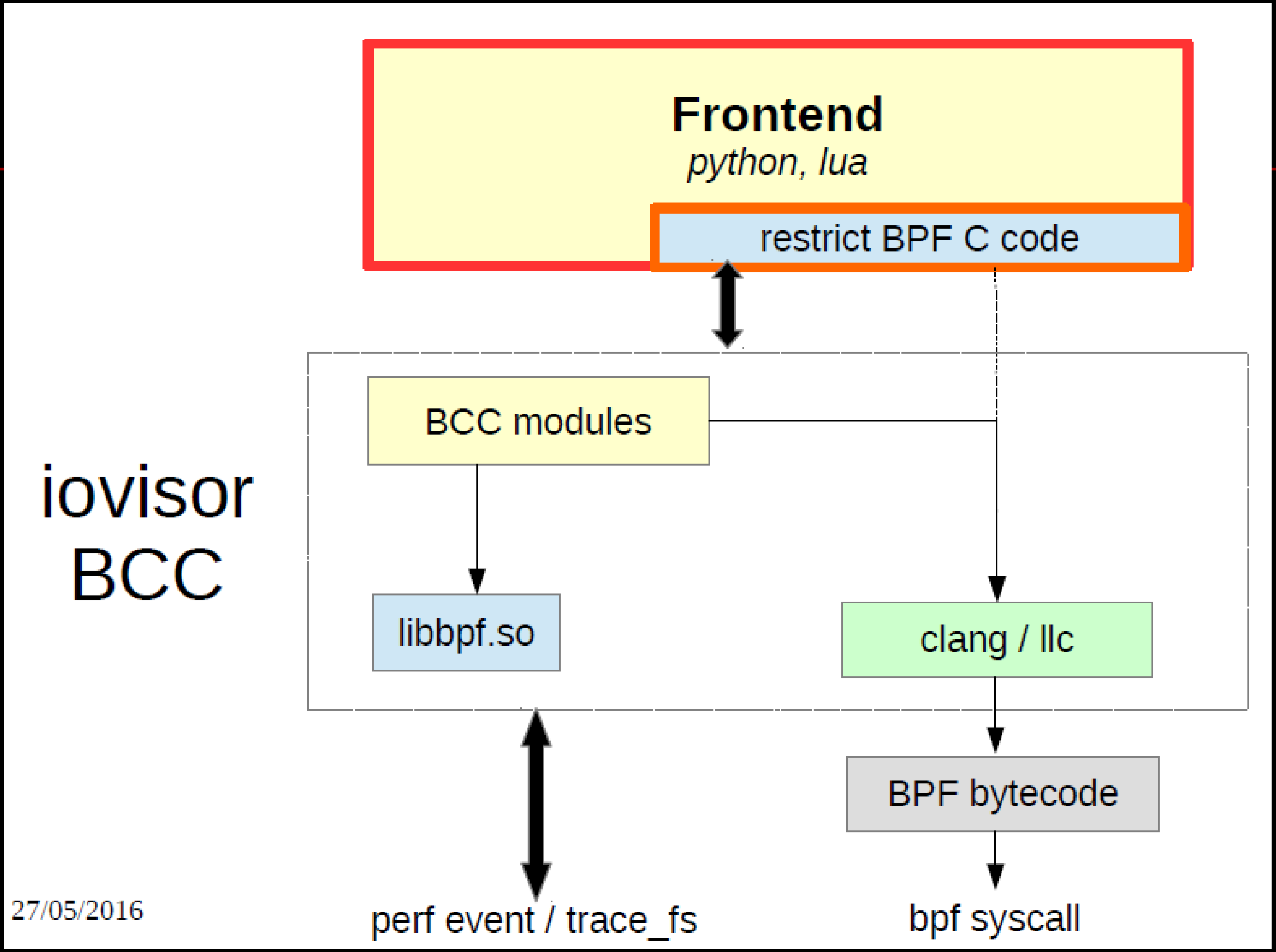
1) BCC (BPF Compiler Collection)

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

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

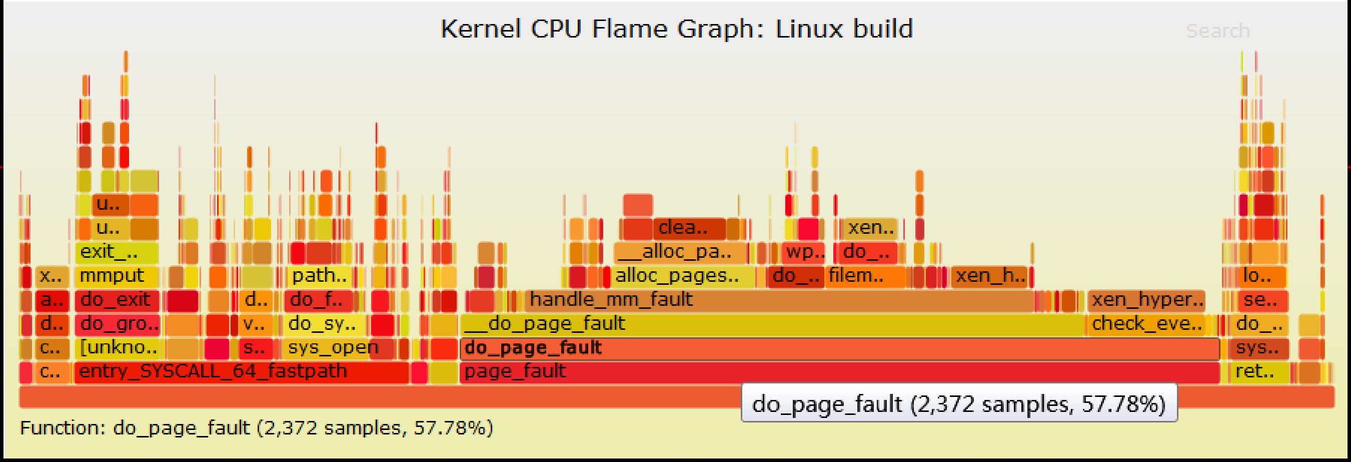
- Compile BPF program from C source
- Attach BPF program to kprobe/uprobe/tracepoint/USDT/socket
- Poll data from BPF program
- Framework for building new tools or one-off scripts
- Contains a P4 compiler for BPF targets
- Additional projects to support Go, Rust, and DTrace-style frontend
- …

Arch



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

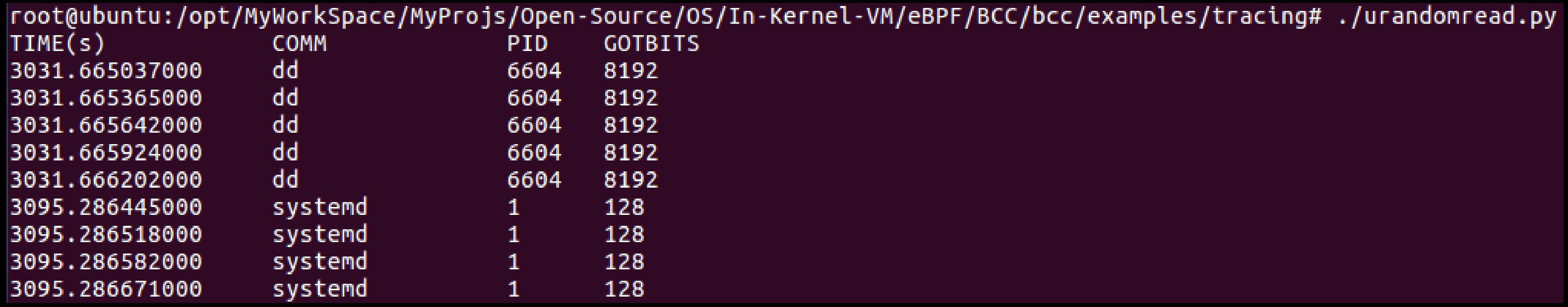
Linux eBPF Flame Graph



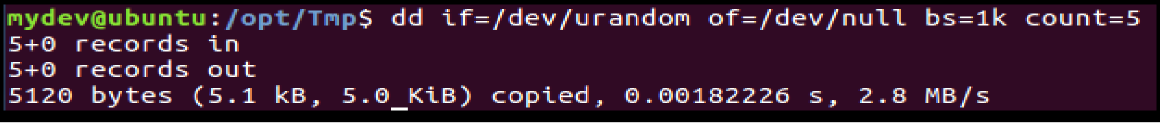
Source: http://www.brendangregg.com/blog/2016-01-20/ebpf-offcpu-flame-graph.html

A Sample

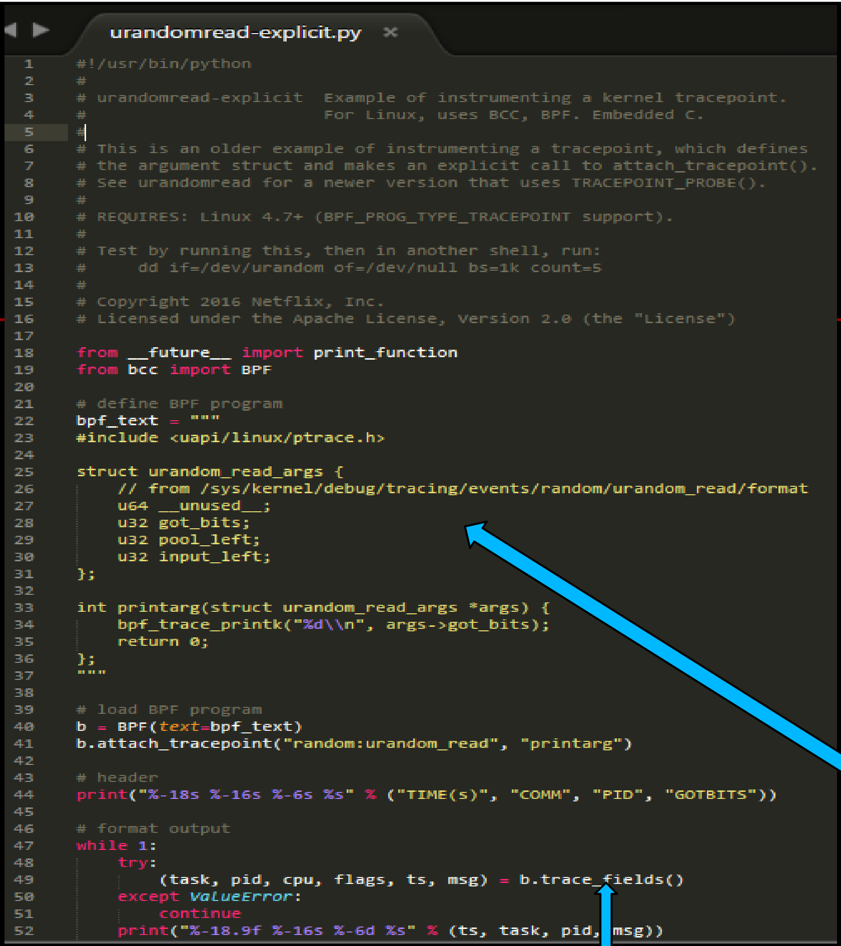
- bcc/examples/tracing/urandomread*.*



```
root@ubuntu:/opt/MyWorkSpace/MyProjs/Open-Source/OS/In-Kernel-VM/eBPF/BCC/bcc/examples/tracing# ./urandomread.py
TIME(s)            COMM             PID    GOTBITS
3031.665037000     dd               6604   8192
3031.665365000     dd               6604   8192
3031.665642000     dd               6604   8192
3031.665924000     dd               6604   8192
3031.666202000     dd               6604   8192
3095.286445000     systemd          1      128
3095.286518000     systemd          1      128
3095.286582000     systemd          1      128
3095.286671000     systemd          1      128
```

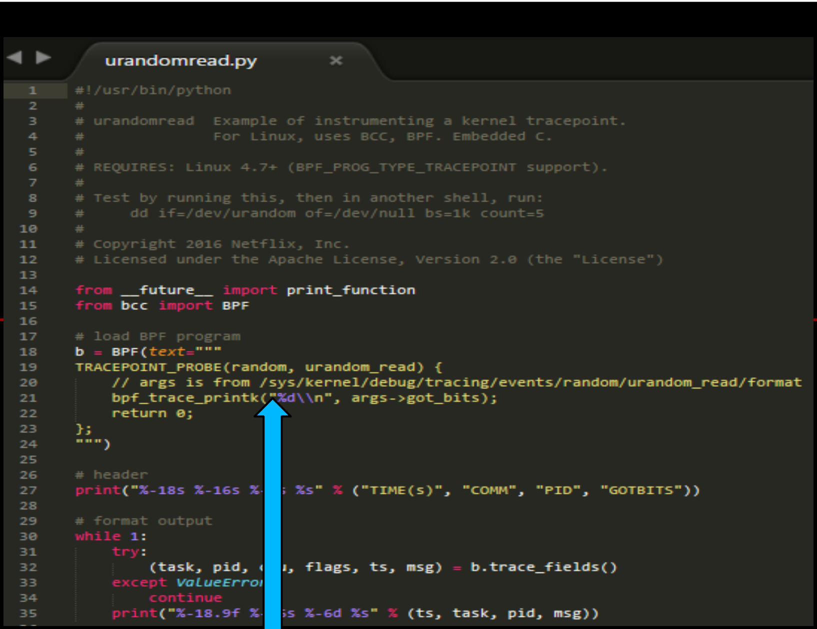


```
mydev@ubuntu:/opt/Tmp$ dd if=/dev/urandom of=/dev/null bs=1k count=5
5+0 records in
5+0 records out
5120 bytes (5.1 kB, 5.0 KiB) copied, 0.00182226 s, 2.8 MB/s
```



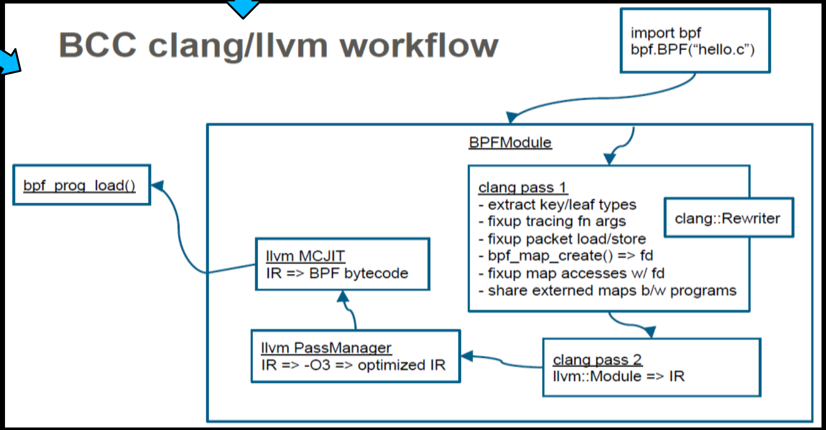
urandomread-explicit.py

```
#!/usr/bin/python
#
# urandomread-explicit  Example of instrumenting a kernel tracepoint.
#                       For Linux, uses BCC, BPF. Embedded C.
#
# This is an older example of instrumenting a tracepoint, which defines
# the argument struct and makes an explicit call to attach_tracepoint().
# See urandomread for a newer version that uses TRACEPOINT_PROBE().
#
# REQUIRES: Linux 4.7+ (BPF_PROG_TYPE_TRACEPOINT support).
#
# Test by running this, then in another shell, run:
#     dd if=/dev/urandom of=/dev/null bs=1k count=5
#
# Copyright 2016 Netflix, Inc.
# Licensed under the Apache License, Version 2.0 (the "License")

from __future__ import print_function
from bcc import BPF

# define BPF program
bpf_text = """
#include <uapi/linux/ptrace.h>

struct urandom_read_args {
    // from /sys/kernel/debug/tracing/events/random/urandom_read/format
    u64 __unused__;
    u32 got_bits;
    u32 pool_left;
    u32 input_left;
};

int printarg(struct urandom_read_args *args) {
    bpf_trace_printk("%d\\n", args->got_bits);
    return 0;
};
"""

# load BPF program
b = BPF(text=bpf_text)
b.attach_tracepoint("random:urandom_read", "printarg")

# header
print("%-18s %-16s %-6s %s" % ("TIME(s)", "COMM", "PID", "GOTBITS"))

# format output
while 1:
    try:
        (task, pid, cpu, flags, ts, msg) = b.trace_fields()
    except ValueError:
        continue
    print("%-18.9f %-16s %-6d %s" % (ts, task, pid, msg))
```



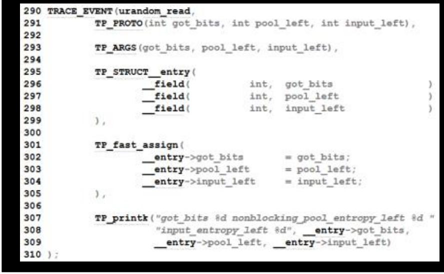
urandomread.py

```
#!/usr/bin/python
#
# urandomread  Example of instrumenting a kernel tracepoint.
#              For Linux, uses BCC, BPF. Embedded C.
#
# REQUIRES: Linux 4.7+ (BPF_PROG_TYPE_TRACEPOINT support).
#
# Test by running this, then in another shell, run:
#     dd if=/dev/urandom of=/dev/null bs=1k count=5
#
# Copyright 2016 Netflix, Inc.
# Licensed under the Apache License, Version 2.0 (the "License")

from __future__ import print_function
from bcc import BPF

# load BPF program
b = BPF(text="""
TRACEPOINT_PROBE(random, urandom_read) {
    // args is from /sys/kernel/debug/tracing/events/random/urandom_read/format
    bpf_trace_printk("%d\\n", args->got_bits);
    return 0;
};
""")

# header
print("%-18s %-16s %-6s %s" % ("TIME(s)", "COMM", "PID", "GOTBITS"))

# format output
while 1:
    try:
        (task, pid, cpu, flags, ts, msg) = b.trace_fields()
    except ValueError:
        continue
    print("%-18.9f %-16s %-6d %s" % (ts, task, pid, msg))
```



BCC clang/llvm workflow

include/trace/events/random.h



```
TRACE_EVENT(urandom_read,
        TP_PROTO(int got_bits, int pool_left, int input_left),

        TP_ARGS(got_bits, pool_left, input_left),

        TP_STRUCT__entry(
                __field(          int,  got_bits             )
                __field(          int,  pool_left            )
                __field(          int,  input_left           )
        ),

        TP_fast_assign(
                __entry->got_bits       = got_bits;
                __entry->pool_left      = pool_left;
                __entry->input_left     = input_left;
        ),

        TP_printk("got_bits %d nonblocking_pool_entropy_left %d "
                  "input_entropy_left %d", __entry->got_bits,
                  __entry->pool_left, __entry->input_left)
);
```

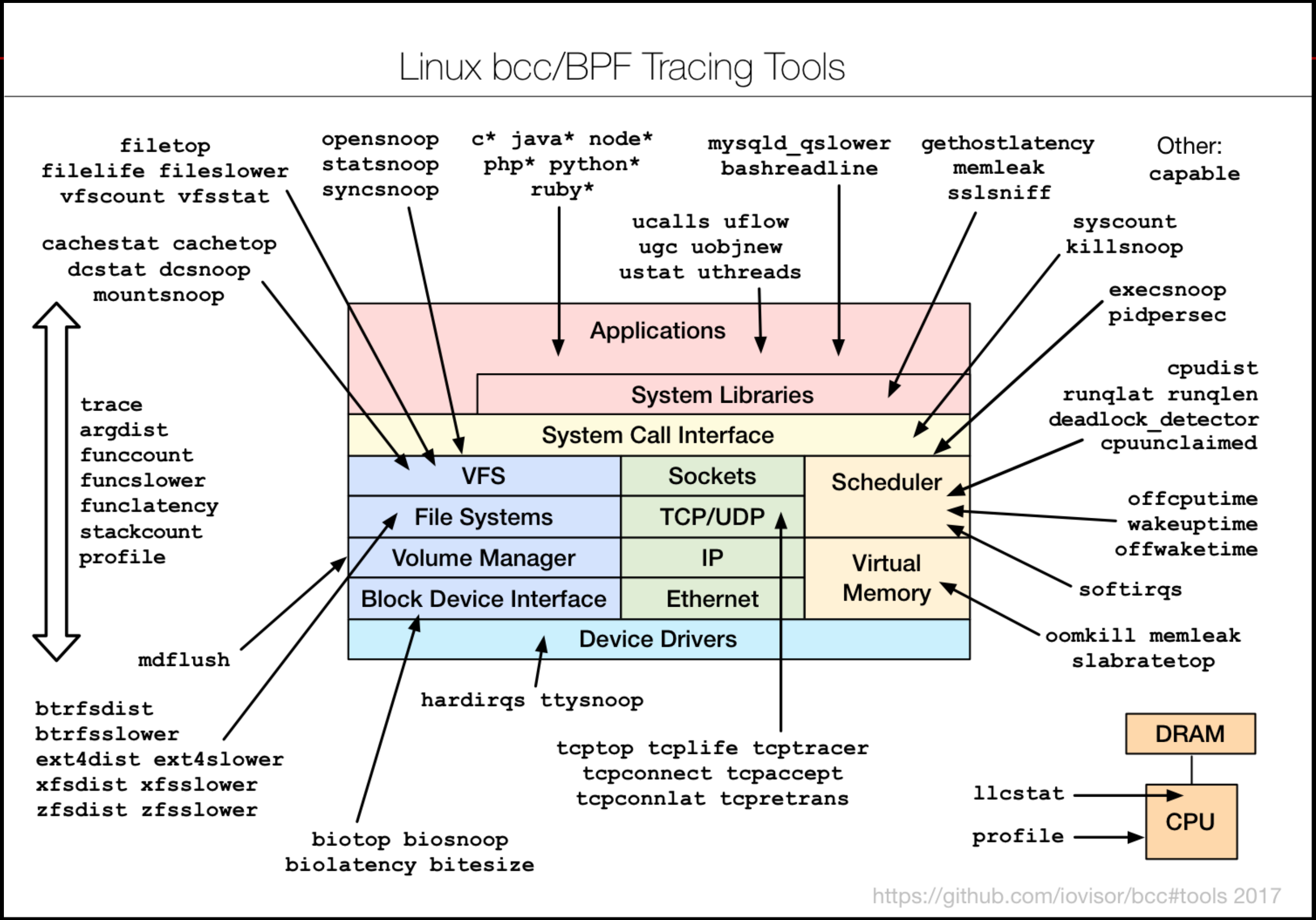
/sys/kernel/debug/tracing/trace_pipe

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

3) Applications

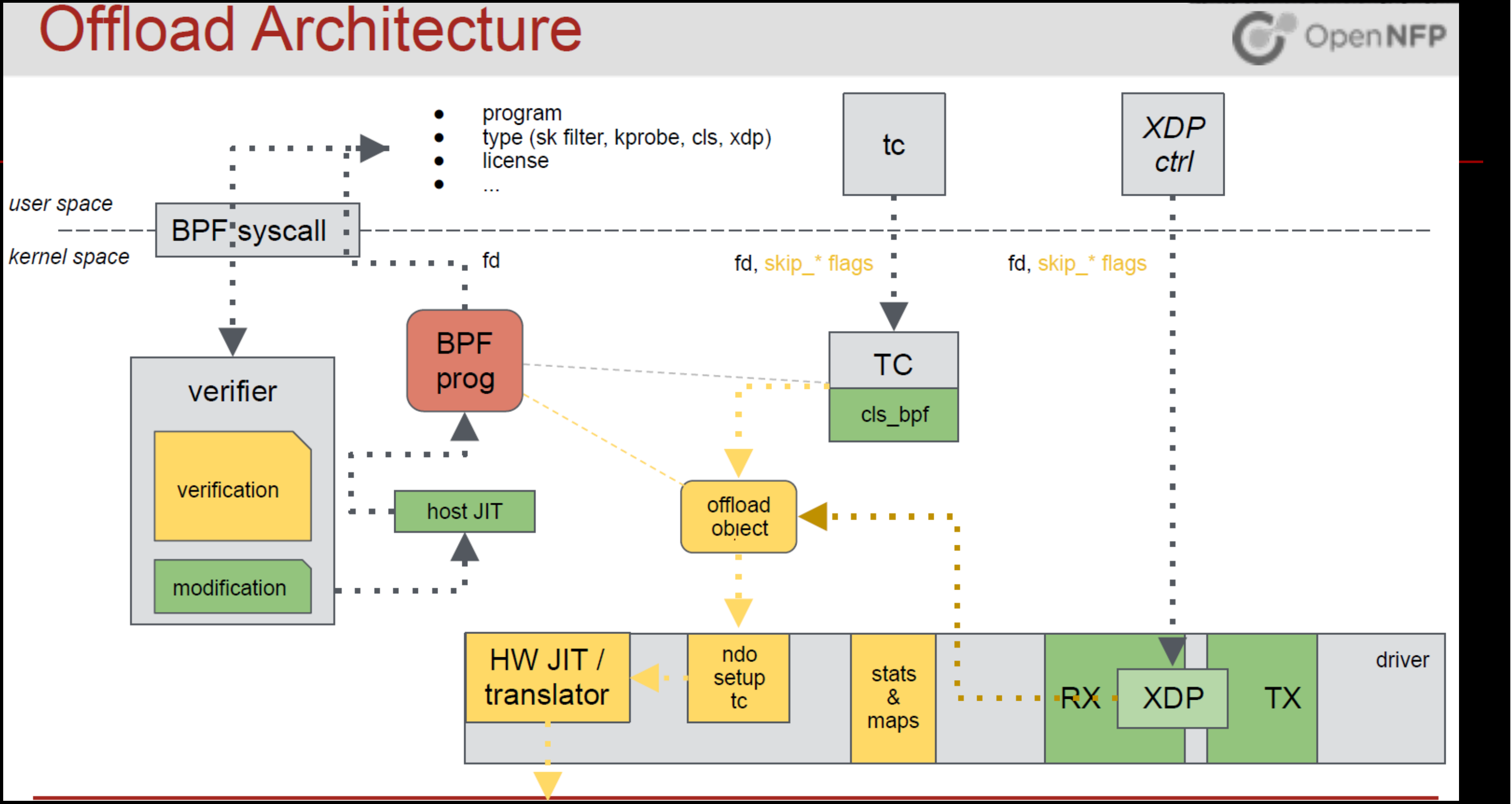
Tuning

- http://www.brendangregg.com/blog/index.html



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

Offloading



Offload Architecture

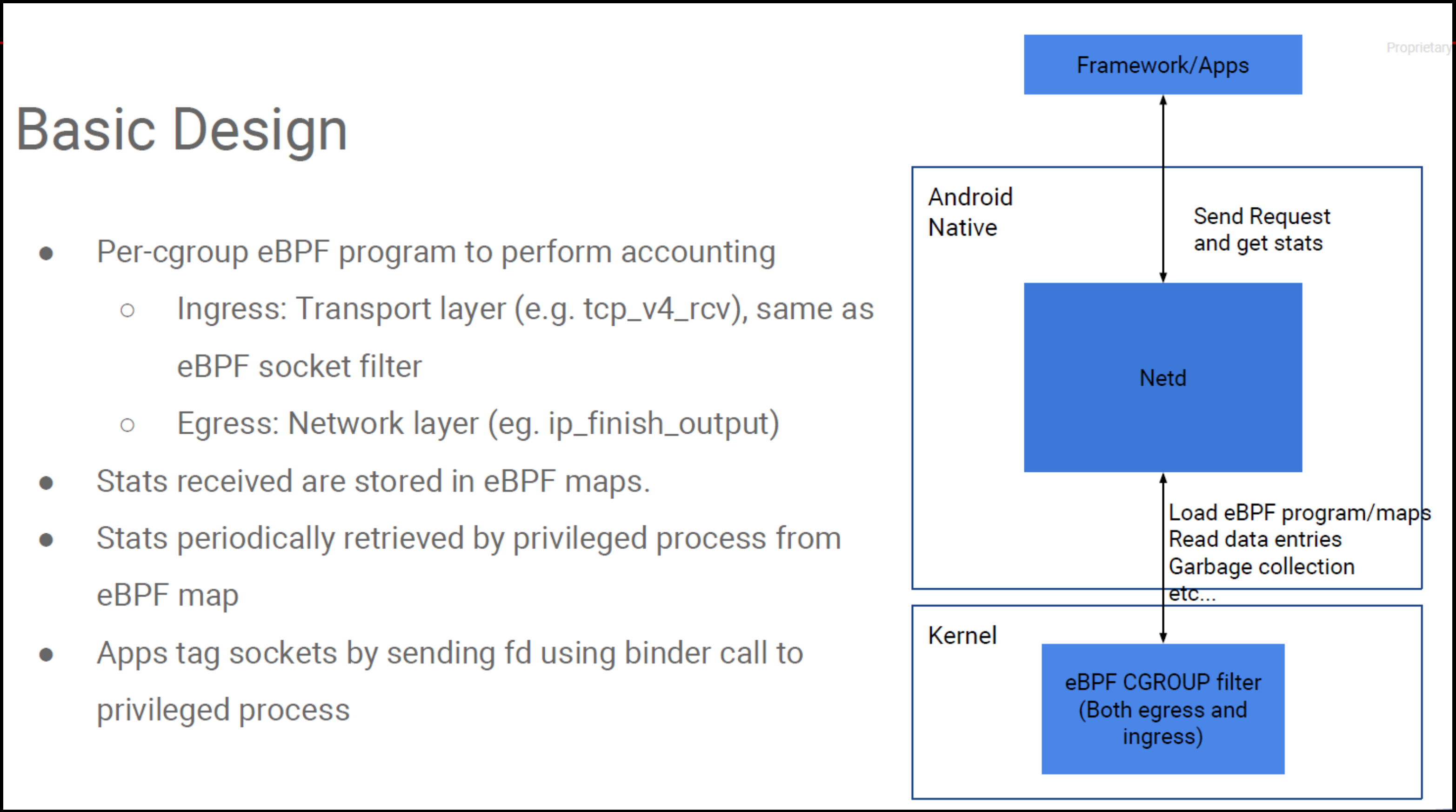
OpenNFP

- program
- type (sk filter, kprobe, cls, xdp)
- license
- ...

Source: https://www.slideshare.net/Open-NFP/transparent-ebpf-offload-playing-nice-with-the-linux-kernel

Netd on Android

- Old: xt_qtaguid module
- New: eBPF cgroup filters for data usage accounting



Proprietary

Basic Design

- Per-cgroup eBPF program to perform accounting
  - Ingress: Transport layer (e.g. tcp_v4_rcv), same as eBPF socket filter
  - Egress: Network layer (eg. ip_finish_output)
- Stats received are stored in eBPF maps.
- Stats periodically retrieved by privileged process from eBPF map
- Apps tag sockets by sending fd using binder call to privileged process

Framework/Apps

Android Native

Send Request and get stats

Netd

Load eBPF program/maps
Read data entries
Garbage collection
etc...

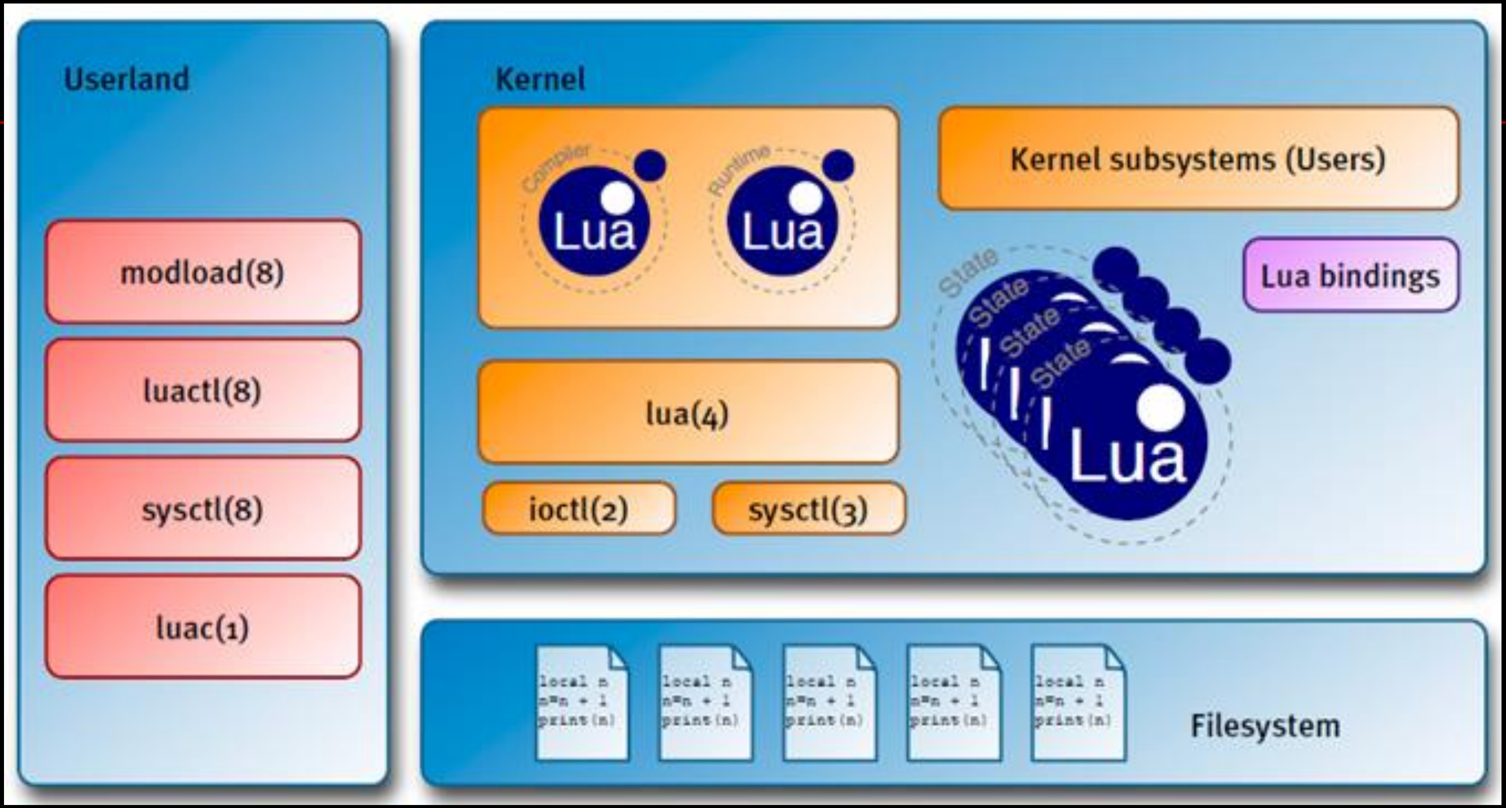
Kernel

eBPF CGROUP filter (Both egress and ingress)

Source: http://www.linuxplumbersconf.org/2017/ocw/proposals/4791

Kernel Development

- NetBSD Kernel scripting with Lua



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

- deliver a higher-level programming environment to the Kernel
- great innovation in OS development

3) 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

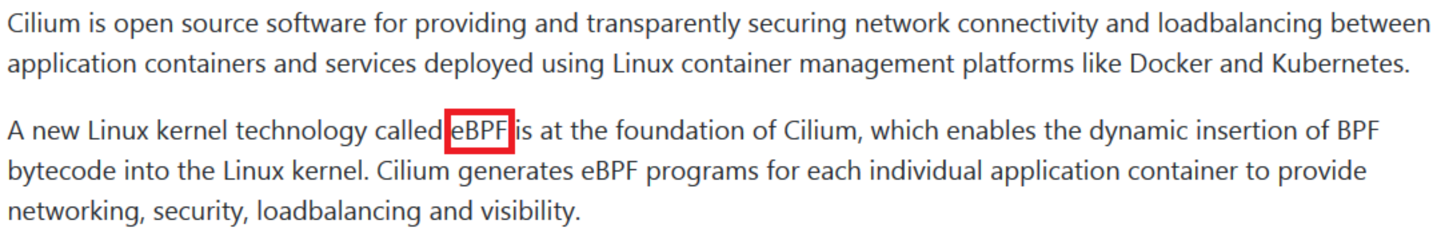
- Up to 512 bytes stack
- Max 4096 instructions per program
- No more than 64 maps
- …

III. Cloud Computing with eBPF

1) Cilium

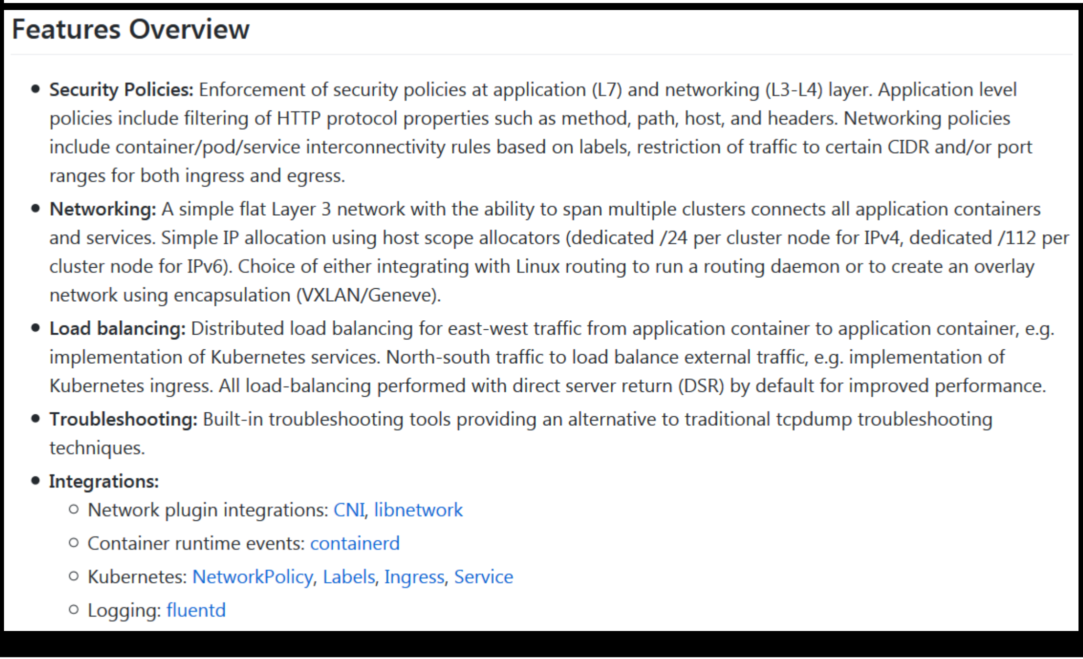
Overview

- https://github.com/cilium/



Cilium is open source software for providing and transparently securing network connectivity and loadbalancing between application containers and services deployed using Linux container management platforms like Docker and Kubernetes.

A new Linux kernel technology called eBPF is at the foundation of Cilium, which enables the dynamic insertion of BPF bytecode into the Linux kernel. Cilium generates eBPF programs for each individual application container to provide networking, security, loadbalancing and visibility.

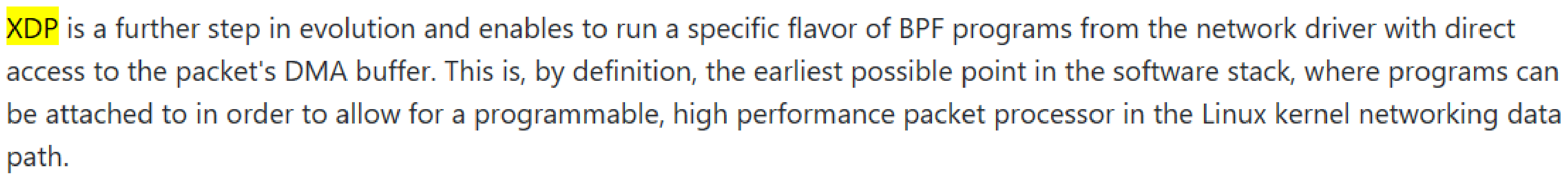


Features Overview

- **Security Policies:** Enforcement of security policies at application (L7) and networking (L3-L4) layer. Application level policies include filtering of HTTP protocol properties such as method, path, host, and headers. Networking policies include container/pod/service interconnectivity rules based on labels, restriction of traffic to certain CIDR and/or port ranges for both ingress and egress.
- **Networking:** A simple flat Layer 3 network with the ability to span multiple clusters connects all application containers and services. Simple IP allocation using host scope allocators (dedicated /24 per cluster node for IPv4, dedicated /112 per cluster node for IPv6). Choice of either integrating with Linux routing to run a routing daemon or to create an overlay network using encapsulation (VXLAN/Geneve).
- **Load balancing:** Distributed load balancing for east-west traffic from application container to application container, e.g. implementation of Kubernetes services. North-south traffic to load balance external traffic, e.g. implementation of Kubernetes ingress. All load-balancing performed with direct server return (DSR) by default for improved performance.
- **Troubleshooting:** Built-in troubleshooting tools providing an alternative to traditional tcpdump troubleshooting techniques.
- **Integrations:**
  - Network plugin integrations: CNI, libnetwork
  - Container runtime events: containerd
  - Kubernetes: NetworkPolicy, Labels, Ingress, Service
  - Logging: fluentd

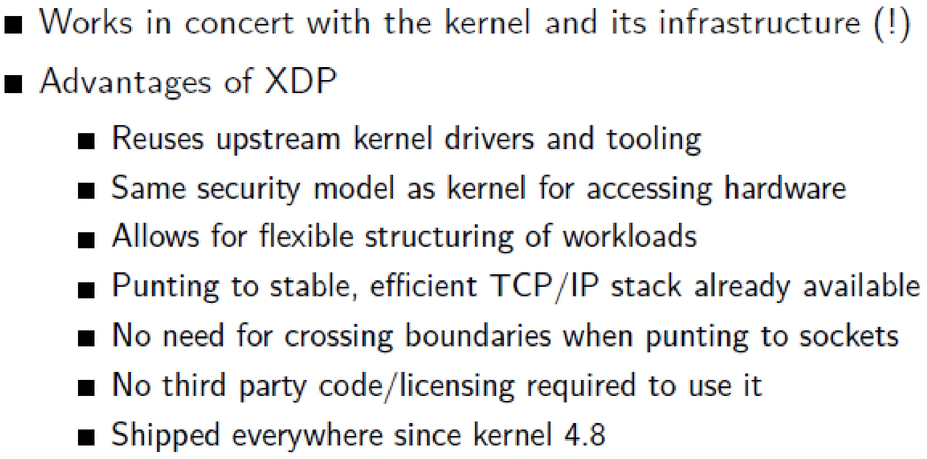
XDP

- https://www.iovisor.org/technology/xdp
- eXpress Data Path
- https://lwn.net/Articles/708087/ //Debating the value of XDP
- Generic hook
- eBPF-based “In-Kernel DPDK”



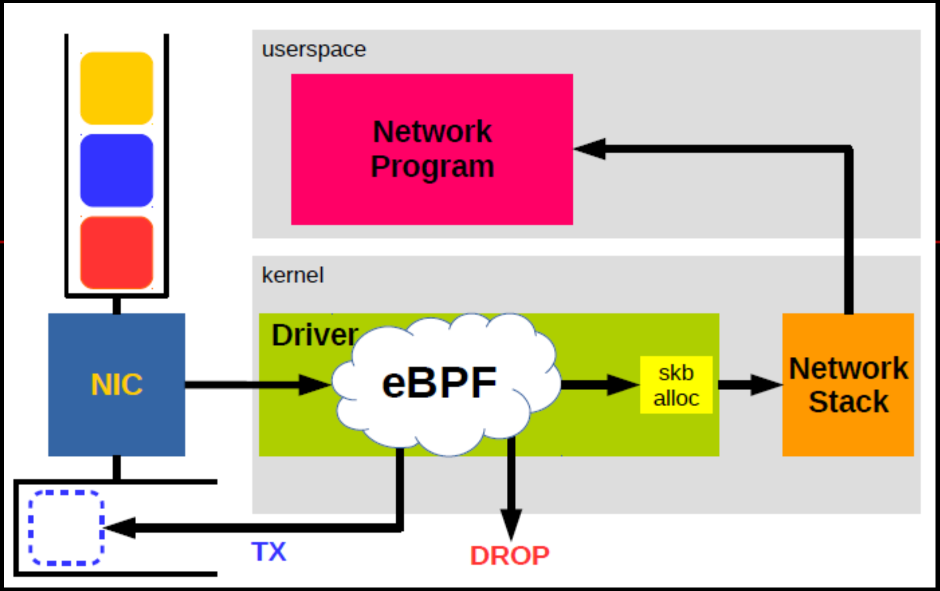
XDP is a further step in evolution and enables to run a specific flavor of BPF programs from the network driver with direct access to the packet's DMA buffer. This is, by definition, the earliest possible point in the software stack, where programs can be attached to in order to allow for a programmable,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

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

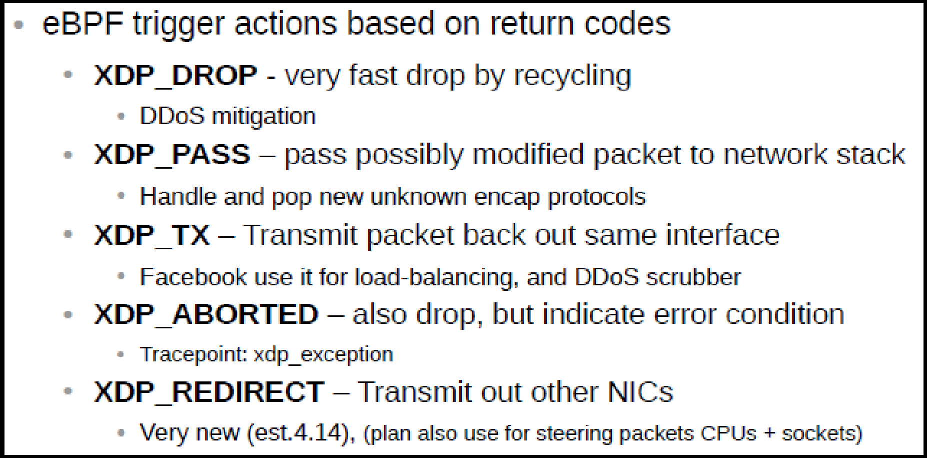


- Works in concert with the kernel and its infrastructure (!)
- Advantages of XDP
  - Reuses upstream kernel drivers and tooling
  - Same security model as kernel for accessing hardware
  - Allows for flexible structuring of workloads
  - Punting to stable, efficient TCP/IP stack already available
  - No need for crossing boundaries when punting to sockets
  - No third party code/licensing required to use it
  - Shipped everywhere since kernel 4.8

Source: https://ossna2017.sched.com/event/BCsg/making-the-kernels-networking-data-path-programmable-with-bpf-and-xdp-daniel-borkmann-covalent



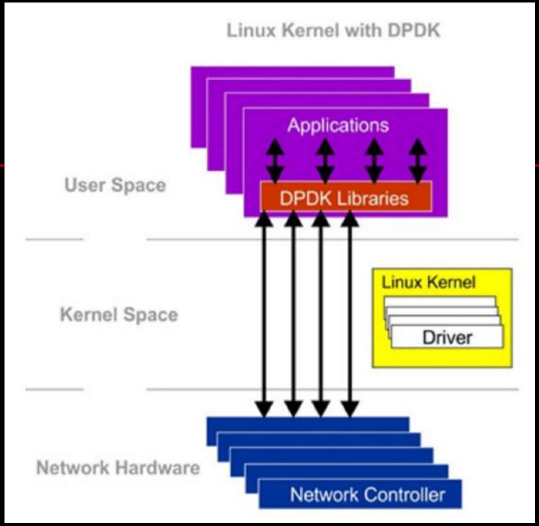
Source: https://www.slideshare.net/lcplcp1/xdp-and-ebpfmaps



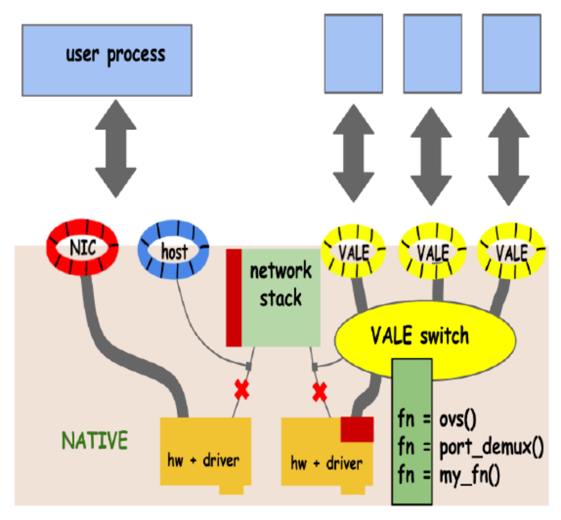
- eBPF trigger actions based on return codes
  - **XDP_DROP** - very fast drop by recycling
    - DDoS mitigation
  - **XDP_PASS** – pass possibly modified packet to network stack
    - Handle and pop new unknown encap protocols
  - **XDP_TX** – Transmit packet back out same interface
    - Facebook 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

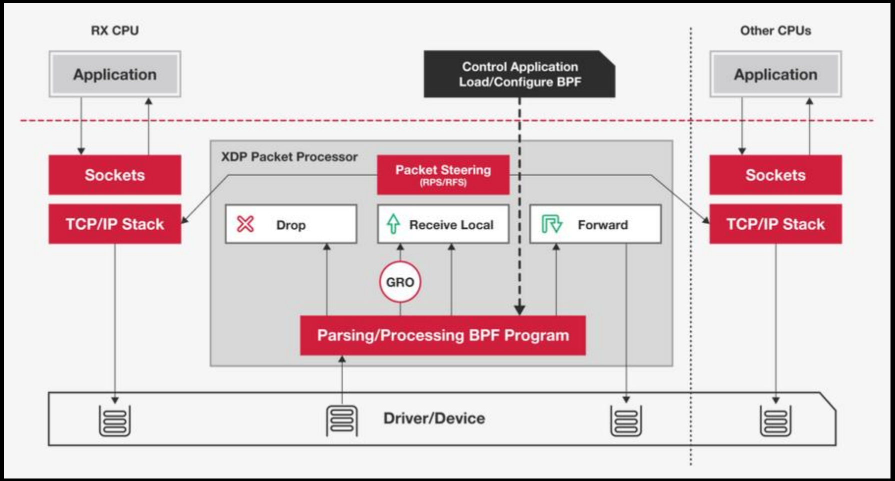
Comparision



DPDK



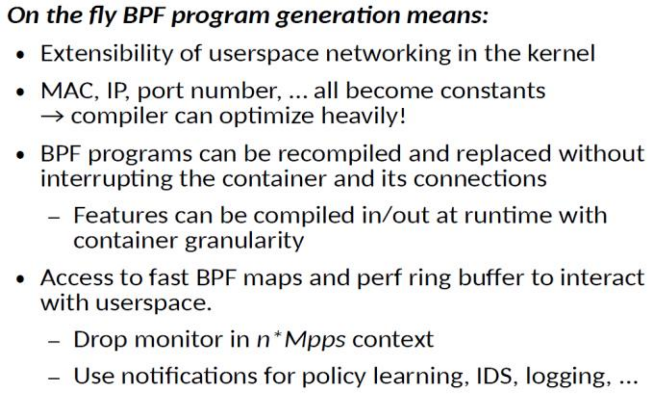
NetMap



XDP

eBPF Code Generation at Container Startup

- Generate networking code at container startup, and tailored to each individual container

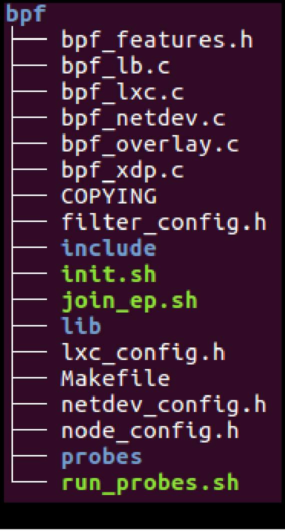


On the fly BPF program generation means:

- Extensibility of userspace networking in the kernel
- MAC, IP, port number, ... all become constants
  → compiler can optimize heavily!
- BPF programs can be recompiled and replaced without interrupting the container and its connections
  - Features can be compiled in/out at runtime with container granularity
- Access to fast BPF maps and perf ring buffer to interact with userspace.
  - Drop monitor in *n*Mpps* context
  - Use notifications for policy learning, IDS, logging, ...

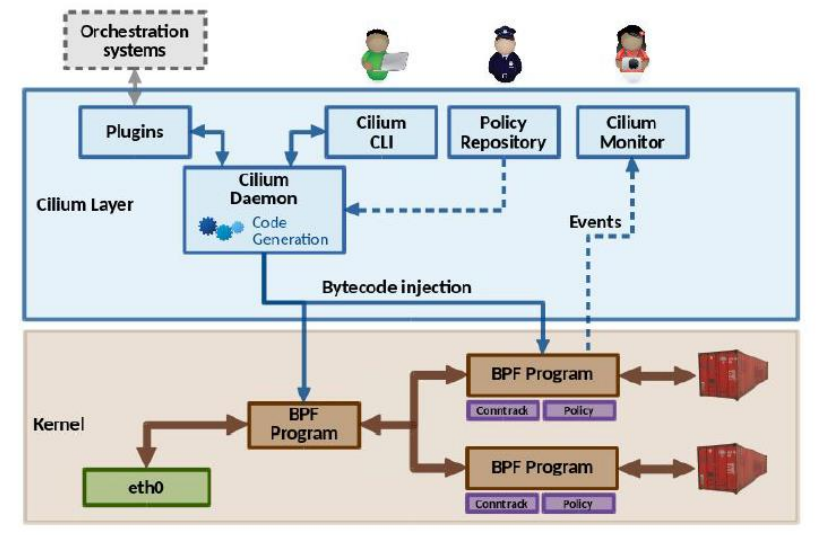
Source: “Cilium: Fast IPv6 Container Networking with BPF and XDP” LinuxCon 2016, Toronto

- $CILIUM_SRC/bpf

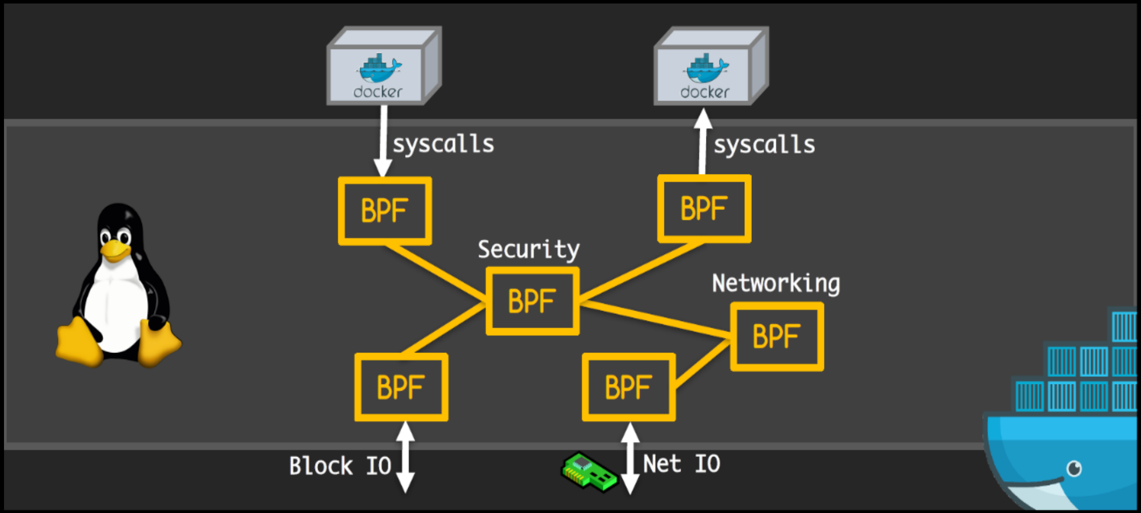


```
bpf
├── bpf_features.h
├── bpf_lb.c
├── bpf_lxc.c
├── bpf_netdev.c
├── bpf_overlay.c
├── bpf_xdp.c
├── COPYING
├── filter_config.h
├── include
├── init.sh
├── join_ep.sh
├── lib
├── lxc_config.h
├── Makefile
├── netdev_config.h
├── node_config.h
├── probes
└── run_probes.sh
```

- ***Arch***



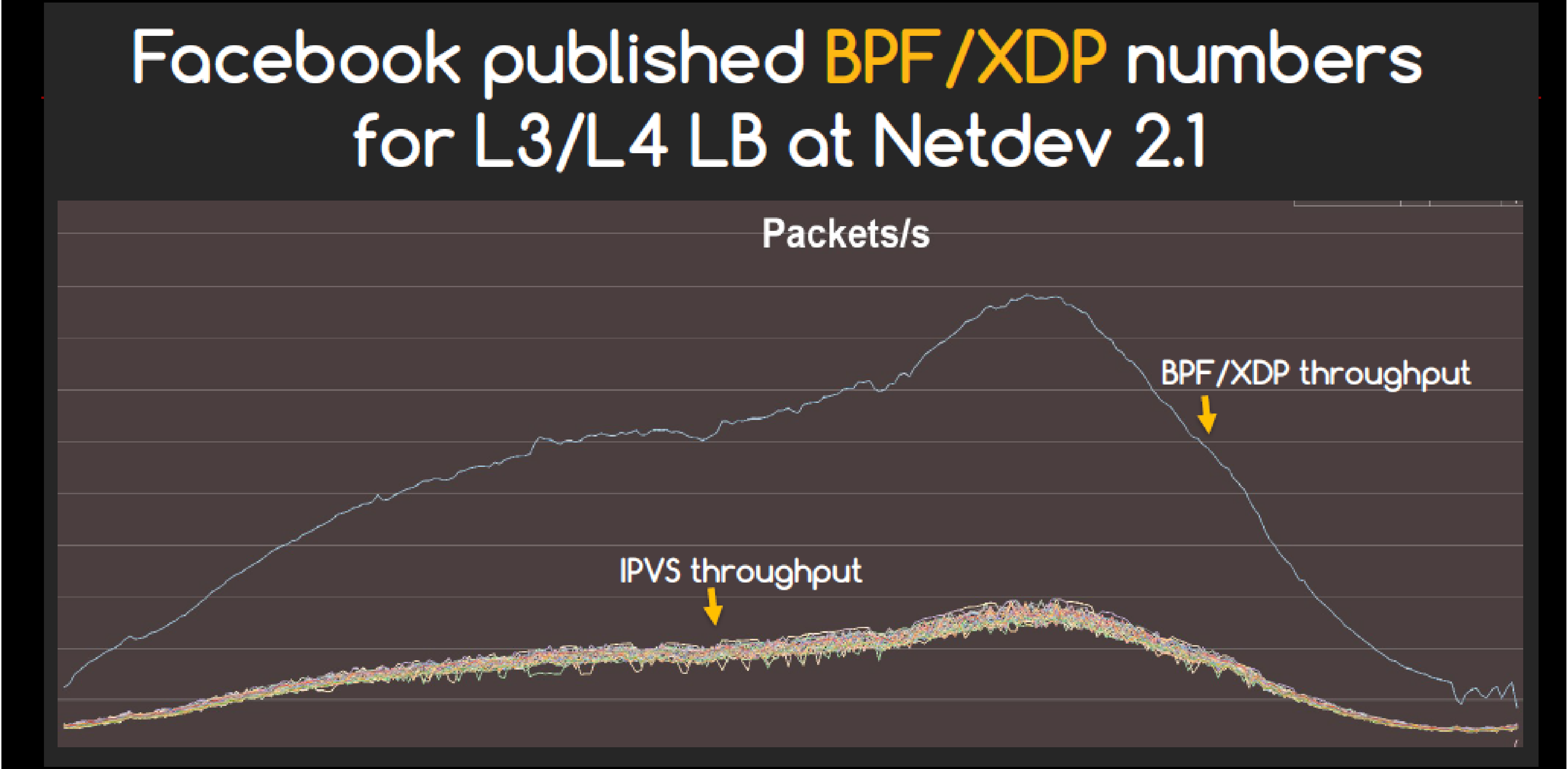
Source: https://www.slideshare.net/ThomasGraf5/cilium-fast-ipv6-container-networking-with-bpf-and-xdp



Source: https://www.slideshare.net/ThomasGraf5/dockercon-2017-cilium-network-and-application-security-with-bpf-and-xdp

2) Load Balance

- https://www.iovisor.org/

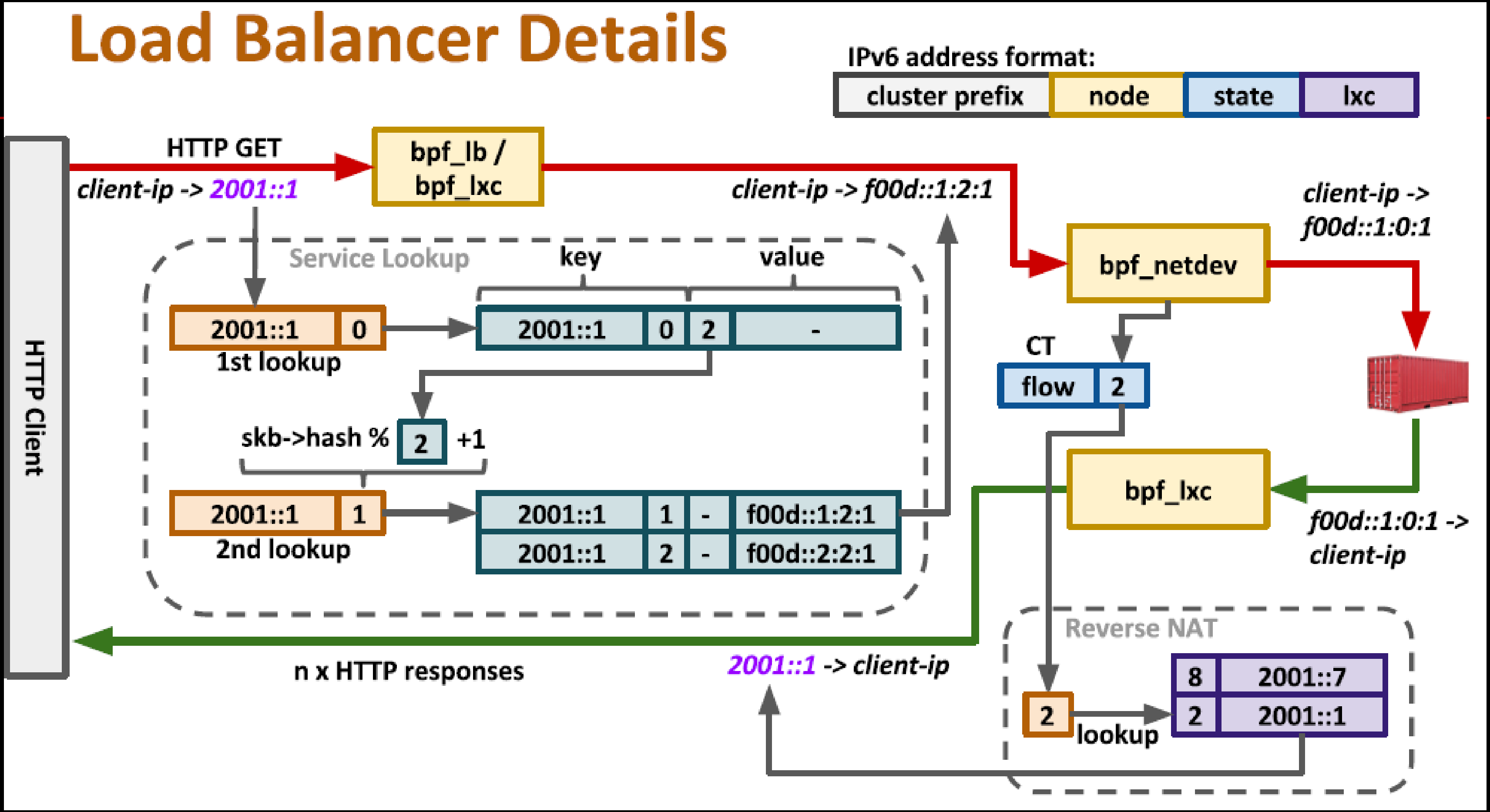


Facebook published BPF/XDP numbers for L3/L4 LB at Netdev 2.1

Source: https://www.slideshare.net/ThomasGraf5/cilium-network-security-for-microservices

LB in Cilium

- scaling policy



Load Balancer Details

IPv6 address format:

cluster prefix	node	state	lxc

HTTP GET

client-ip -> 2001::1

bpf_lb / bpf_lxc

client-ip -> f00d::1:2:1

client-ip -> f00d::1:0:1

HTTP Client

Service Lookup

key

value

2001::1	0

1st lookup

2001::1	0	2	-

skb->hash % 2 +1

2001::1	1

2nd lookup

2001::1	1	-	f00d::1:2:1
2001::1	2	-	f00d::2:2:1

bpf_netdev

CT

flow	2

bpf_lxc

f00d::1:0:1 -> client-ip

n x HTTP responses

2001::1 -> client-ip

Reverse NAT

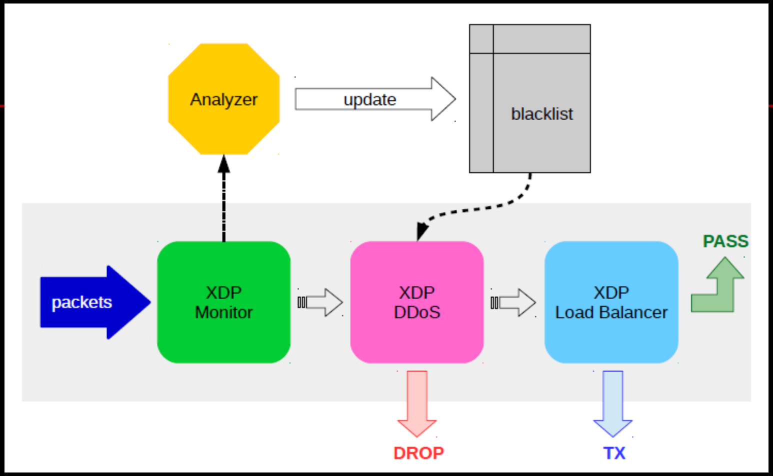
2 lookup

8	2001::7
2	2001::1

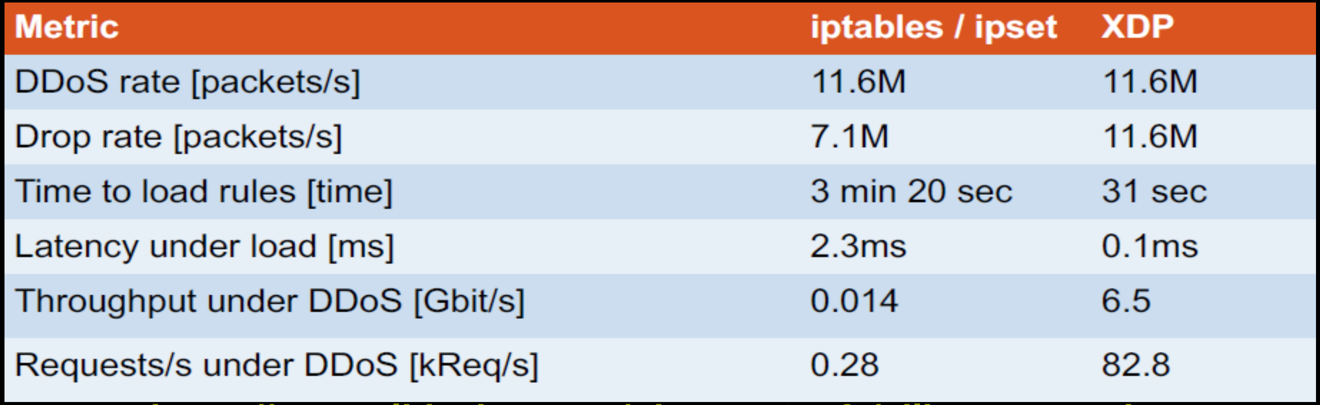
Source: https://www.slideshare.net/ThomasGraf5/clium-container-networking-with-bpf-xdp

3) Security

DDoS Protection



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

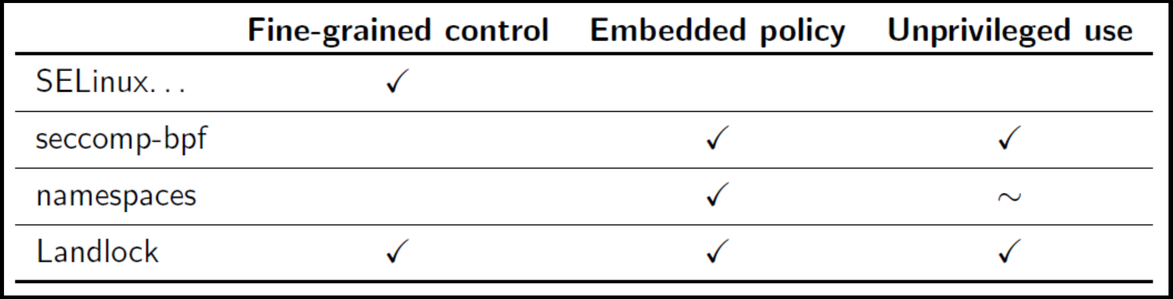


Metric	iptables / ipset	XDP
DDoS rate [packets/s]	11.6M	11.6M
Drop rate [packets/s]	7.1M	11.6M
Time to load rules [time]	3 min 20 sec	31 sec
Latency under load [ms]	2.3ms	0.1ms
Throughput under DDoS [Gbit/s]	0.014	6.5
Requests/s under DDoS [kReq/s]	0.28	82.8

Source: https://www.slideshare.net/ThomasGraf5/cilium-network-security-for-microservices

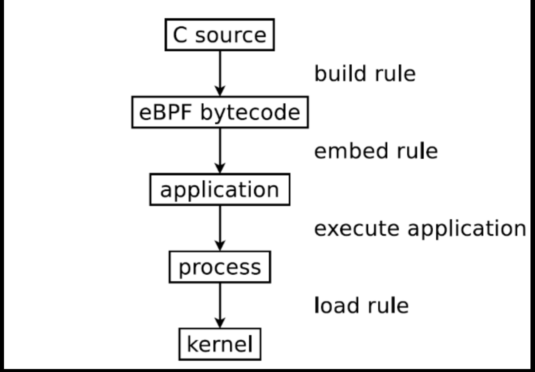
LandLock

- Linux Security Modules
- https://en.wikipedia.org/wiki/Linux_Security_Modules
- https://www.kernel.org/doc/Documentation/security/LSM.txt
- https://landlock.io/
- https://github.com/landlock-lsm/linux/commits/landlock-v7
-



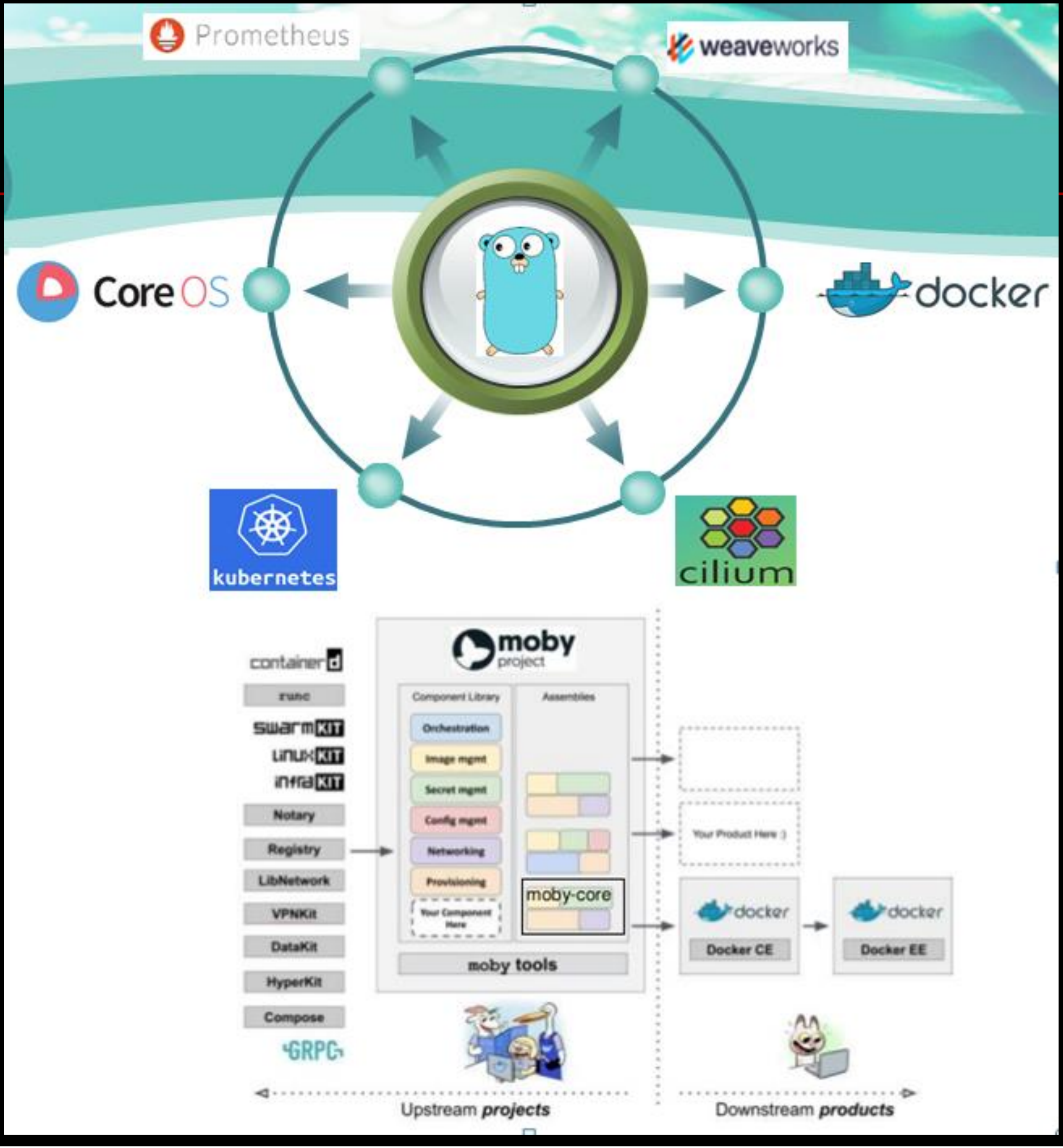
	Fine-grained control	Embedded policy	Unprivileged use
SELinux. . .	✓		
seccomp-bpf		✓	✓
namespaces		✓	∼
Landlock	✓	✓	✓

Life cycle of a Landlock rule



Source: http://events.linuxfoundation.org/sites/events/files/slides/2017-09-14_landlock-lss.pdf

4) Go-based Cloud Ecosystem



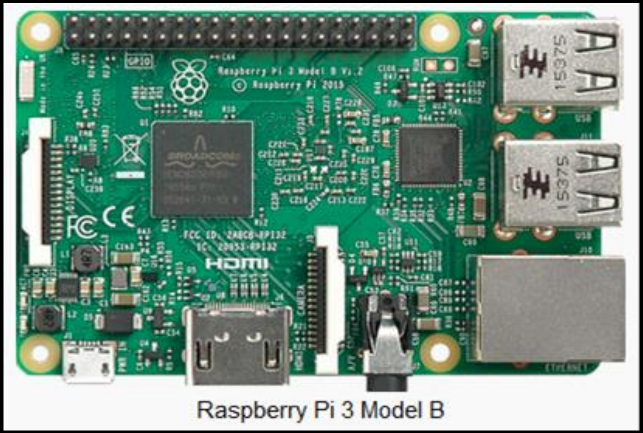
IV. eBPF on ARM

1) RPi3

- https://en.wikipedia.org/wiki/Raspberry_Pi
- https://www.raspberrypi.org/

RPi3 Model B

-

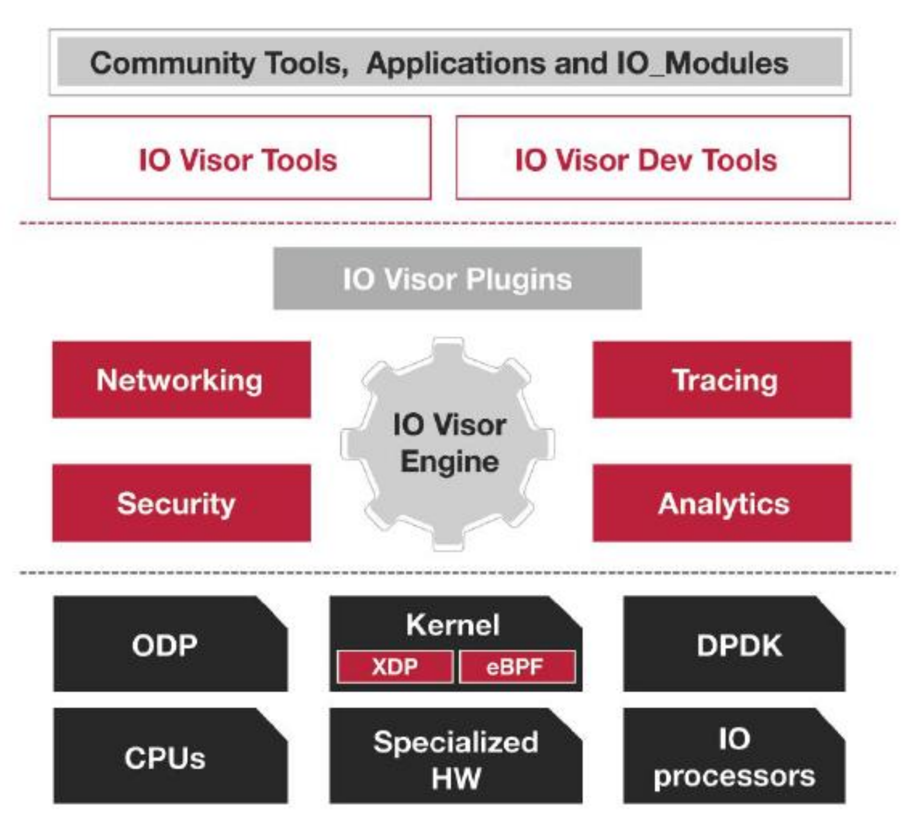


Limitations

1) 1.2 GHz 64-bit quad-core ARM Cortex-A53, 1GB LPDDR2 RAM @900MHz…
2) Official release (Raspbian with Linux Kernel 4.4 currently) does not support AArch64

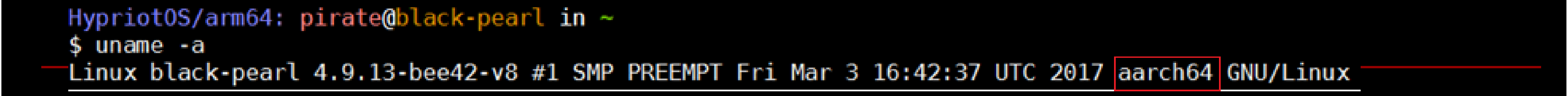
2) IO Visor

- https://www.iovisor.org/
- Advancing In-Kernel IO Virtualization By Enabling Programmable Data Planes With Extensibility, Flexibility and High Performance
- eBPF-based



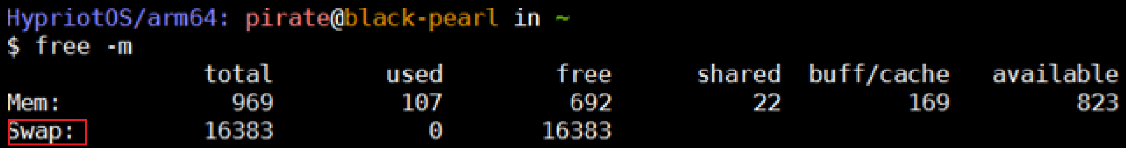
HypriotOS-RPi64

- https://blog.hypriot.com/post/building-a-64bit-docker-os-for-rpi3/
- https://github.com/dieterreuter/workshop-raspberrypi-64bit-os

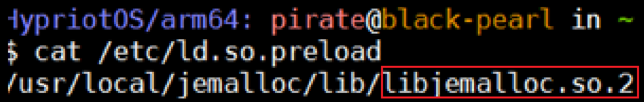


```
HypriotOS/arm64: pirate@black-pearl in ~
$ uname -a
Linux black-pearl 4.9.13-bee42-v8 #1 SMP PREEMPT Fri Mar 3 16:42:37 UTC 2017 aarch64 GNU/Linux
```

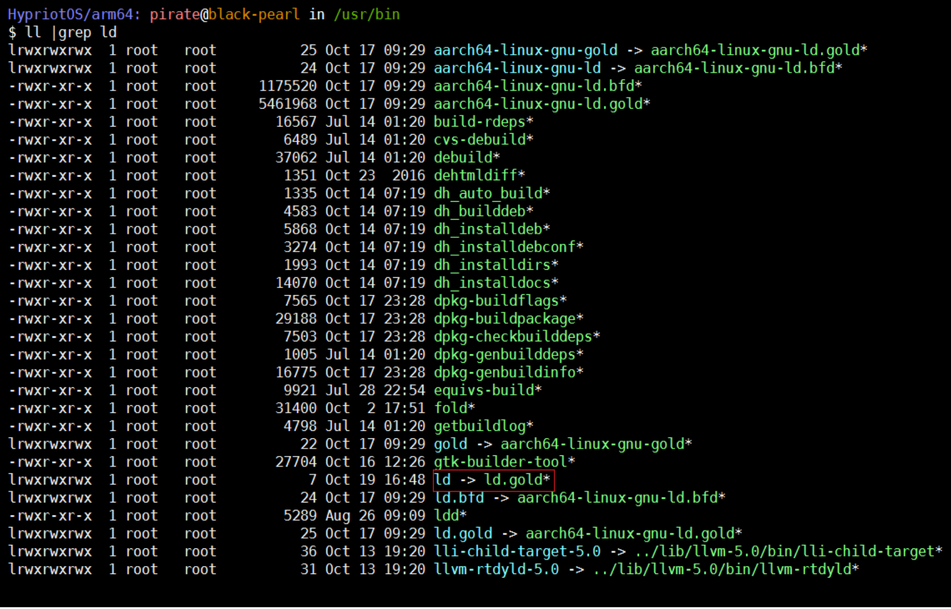
- **BCC on RPi3**



```
HypriotOS/arm64: pirate@black-pearl in ~
$ free -m
              total        used        free      shared  buff/cache   available
Mem:            969         107         692          22         169         823
Swap:         16383           0       16383
```

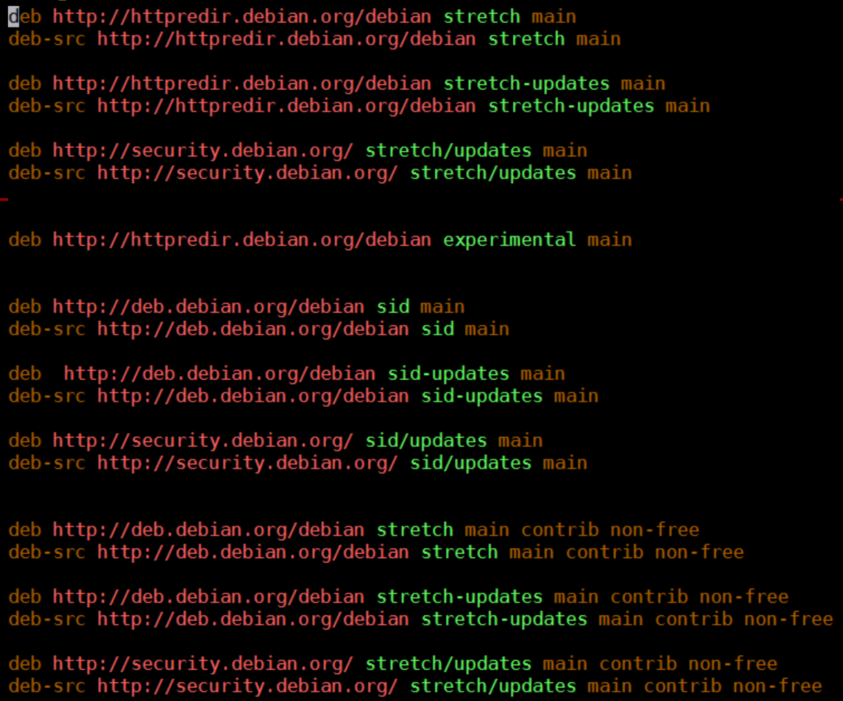


```
HypriotOS/arm64: pirate@black-pearl in ~
$ cat /etc/ld.so.preload
/usr/local/jemalloc/lib/libjemalloc.so.2
```



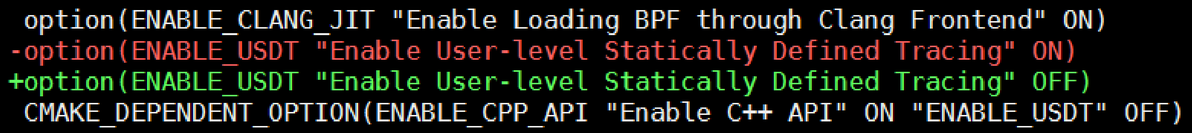
```
HypriotOS/arm64: pirate@black-pearl in /usr/bin
$ ll |grep ld
lrwxrwxrwx  1 root   root         25 Oct 17 09:29 aarch64-linux-gnu-gold -> aarch64-linux-gnu-ld.gold*
lrwxrwxrwx  1 root   root         24 Oct 17 09:29 aarch64-linux-gnu-ld -> aarch64-linux-gnu-ld.bfd*
-rwxr-xr-x  1 root   root    1175520 Oct 17 09:29 aarch64-linux-gnu-ld.bfd*
-rwxr-xr-x  1 root   root    5461968 Oct 17 09:29 aarch64-linux-gnu-ld.gold*
-rwxr-xr-x  1 root   root      16567 Jul 14 01:20 build-rdeps*
-rwxr-xr-x  1 root   root       6489 Jul 14 01:20 cvs-debuild*
-rwxr-xr-x  1 root   root      37062 Jul 14 01:20 debuild*
-rwxr-xr-x  1 root   root       1351 Oct 23  2016 dehtmldiff*
-rwxr-xr-x  1 root   root       1335 Oct 14 07:19 dh_auto_build*
-rwxr-xr-x  1 root   root       4583 Oct 14 07:19 dh_builddeb*
-rwxr-xr-x  1 root   root       5868 Oct 14 07:19 dh_installdeb*
-rwxr-xr-x  1 root   root       3274 Oct 14 07:19 dh_installdebconf*
-rwxr-xr-x  1 root   root       1993 Oct 14 07:19 dh_installdirs*
-rwxr-xr-x  1 root   root      14070 Oct 14 07:19 dh_installdocs*
-rwxr-xr-x  1 root   root       7565 Oct 17 23:28 dpkg-buildflags*
-rwxr-xr-x  1 root   root      29188 Oct 17 23:28 dpkg-buildpackage*
-rwxr-xr-x  1 root   root       7503 Oct 17 23:28 dpkg-checkbuilddeps*
-rwxr-xr-x  1 root   root       1005 Jul 14 01:20 dpkg-genbuilddeps*
-rwxr-xr-x  1 root   root      16775 Oct 17 23:28 dpkg-genbuildinfo*
-rwxr-xr-x  1 root   root       9921 Jul 28 22:54 equivs-build*
-rwxr-xr-x  1 root   root      31400 Oct  2 17:51 fold*
-rwxr-xr-x  1 root   root       4798 Jul 14 01:20 getbuildlog*
lrwxrwxrwx  1 root   root         22 Oct 17 09:29 gold -> aarch64-linux-gnu-gold*
-rwxr-xr-x  1 root   root      27704 Oct 16 12:26 gtk-builder-tool*
lrwxrwxrwx  1 root   root          7 Oct 19 16:48 ld -> ld.gold*
lrwxrwxrwx  1 root   root         24 Oct 17 09:29 ld.bfd -> aarch64-linux-gnu-ld.bfd*
-rwxr-xr-x  1 root   root       5289 Aug 26 09:09 ldd*
lrwxrwxrwx  1 root   root         25 Oct 17 09:29 ld.gold -> aarch64-linux-gnu-ld.gold*
lrwxrwxrwx  1 root   root         36 Oct 13 19:20 lli-child-target-5.0 -> ../lib/llvm-5.0/bin/lli-child-target*
lrwxrwxrwx  1 root   root         31 Oct 13 19:20 llvm-rtdyld-5.0 -> ../lib/llvm-5.0/bin/llvm-rtdyld*
```

repos



```
deb http://httpredir.debian.org/debian stretch main
deb-src http://httpredir.debian.org/debian stretch main

deb http://httpredir.debian.org/debian stretch-updates main
deb-src http://httpredir.debian.org/debian stretch-updates main

deb http://security.debian.org/ stretch/updates main
deb-src http://security.debian.org/ stretch/updates main


deb http://httpredir.debian.org/debian experimental main


deb http://deb.debian.org/debian sid main
deb-src http://deb.debian.org/debian sid main

deb  http://deb.debian.org/debian sid-updates main
deb-src http://deb.debian.org/debian sid-updates main

deb http://security.debian.org/ sid/updates main
deb-src http://security.debian.org/ sid/updates main


deb http://deb.debian.org/debian stretch main contrib non-free
deb-src http://deb.debian.org/debian stretch main contrib non-free

deb http://deb.debian.org/debian stretch-updates main contrib non-free
deb-src http://deb.debian.org/debian stretch-updates main contrib non-free

deb http://security.debian.org/ stretch/updates main contrib non-free
deb-src http://security.debian.org/ stretch/updates main contrib non-free
```

patch

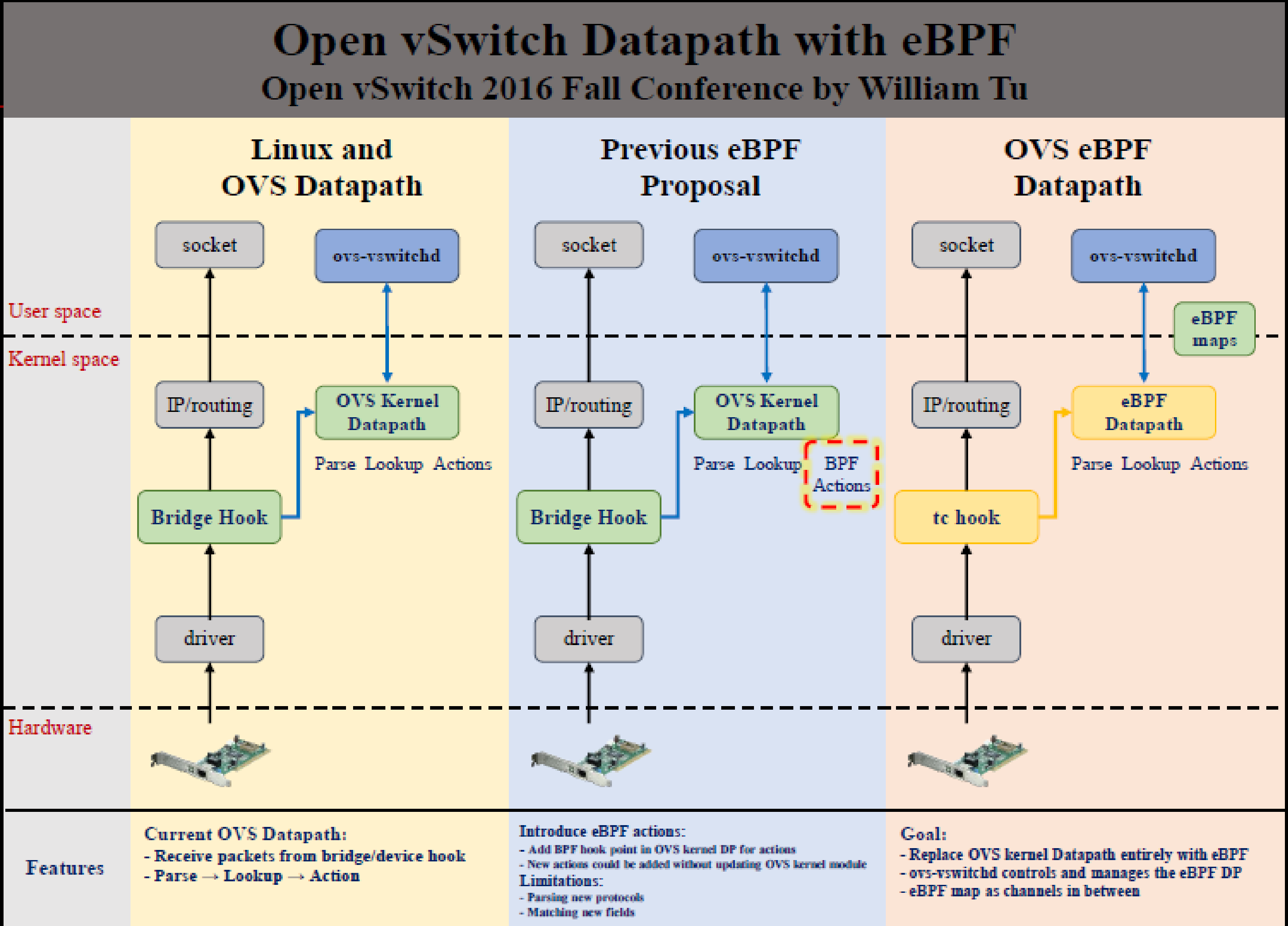


```
 option(ENABLE_CLANG_JIT "Enable Loading BPF through Clang Frontend" ON)
-option(ENABLE_USDT "Enable User-level Statically Defined Tracing" ON)
+option(ENABLE_USDT "Enable User-level Statically Defined Tracing" OFF)
 CMAKE_DEPENDENT_OPTION(ENABLE_CPP_API "Enable C++ API" ON "ENABLE_USDT" OFF)
```

- **build BCC(master branch) by gcc 7.2.0 + jemalloc 5.0.1 + ld.gold ~ 43 minutes**

OVS

- http://openvswitch.org/



Open vSwitch Datapath with eBPF

Open vSwitch 2016 Fall Conference by William Tu

Features

Current OVS Datapath:
- Receive packets from bridge/device hook
- Parse → Lookup → Action

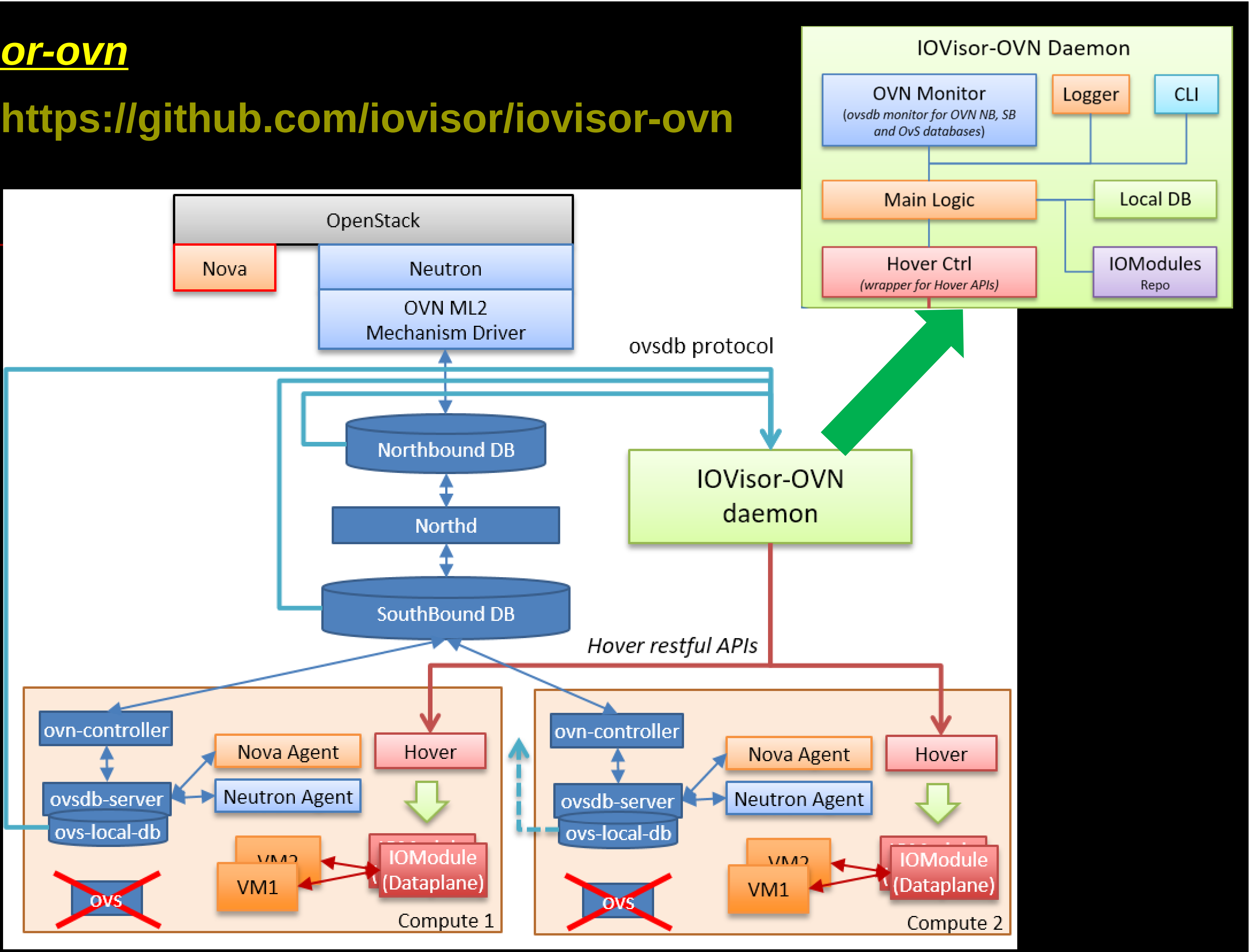
Introduce eBPF actions:
- Add BPF hook point in OVS kernel DP for actions
- New actions could be added without updating OVS kernel module

Limitations:
- Parsing new protocols
- Matching new fields

Goal:
- Replace OVS kernel Datapath entirely with eBPF
- ovs-vswitchd controls and manages the eBPF DP
- eBPF map as channels in between

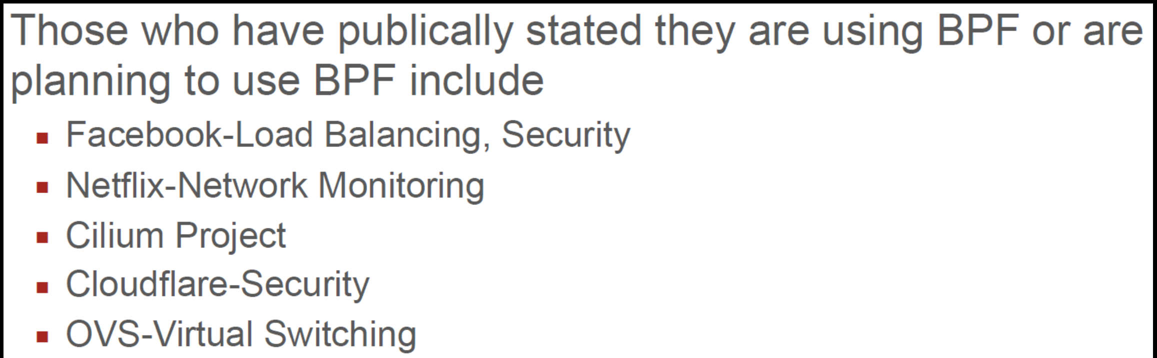
iovisor-ovn

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



V. Wrap-up

- A wide range of applications


Those who have publically stated they are using BPF or are planning to use BPF include

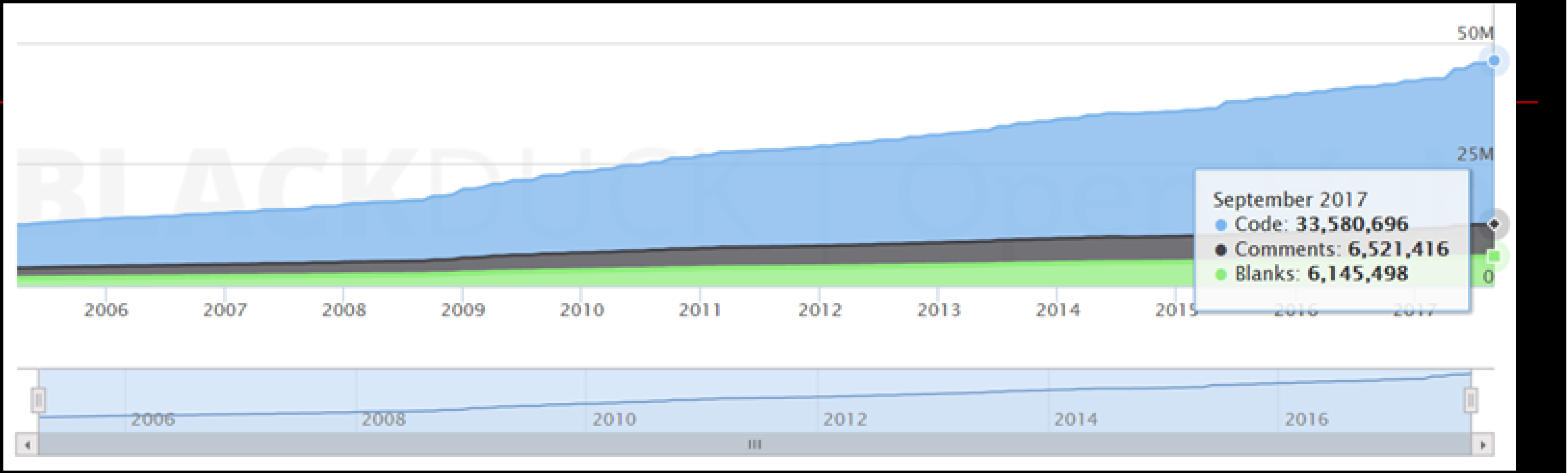
- Facebook-Load Balancing, Security
- Netflix-Network Monitoring
- Cilium Project
- Cloudflare-Security
- OVS-Virtual Switching

…

Source: https://www.slideshare.net/Open-NFP/transparent-ebpf-offload-playing-nice-with-the-linux-kernel

- **Polyglot VM**

Changing the way you think about Linux Kernel development:



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

- **User space/Kernel space Repartition & Unifying**

eBPF is sure to play an important role in tomorrow's Linux!

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/Documentation/
- http://man7.org/linux/man-pages/man2/bpf.2.html
- https://www.python.org
- http://llvm.org
- https://en.wikipedia.org/wiki/Just-in-time_compilation
- http://dpdk.org/
- https://www.netbsd.org/gallery/presentations/
- https://www.opennetworking.org/
- https://www.opnfv.org/
- …