

Quick Introduction to eBPF & BCC

Suchakrapani Sharma

Kernel Meetup (Reserved Bit, Pune)



eBPF

Stateful, programmable,
in-kernel decisions for
networking, tracing and security



Berkeley Packet Filter

Classical BPF (cBPF)

- Network packet filtering [McCanne et al. 1993], Seccomp
- Filter Expressions → Bytecode → Interpret*
- Small, in-kernel VM. Register based, switch dispatch interpreter, few instructions

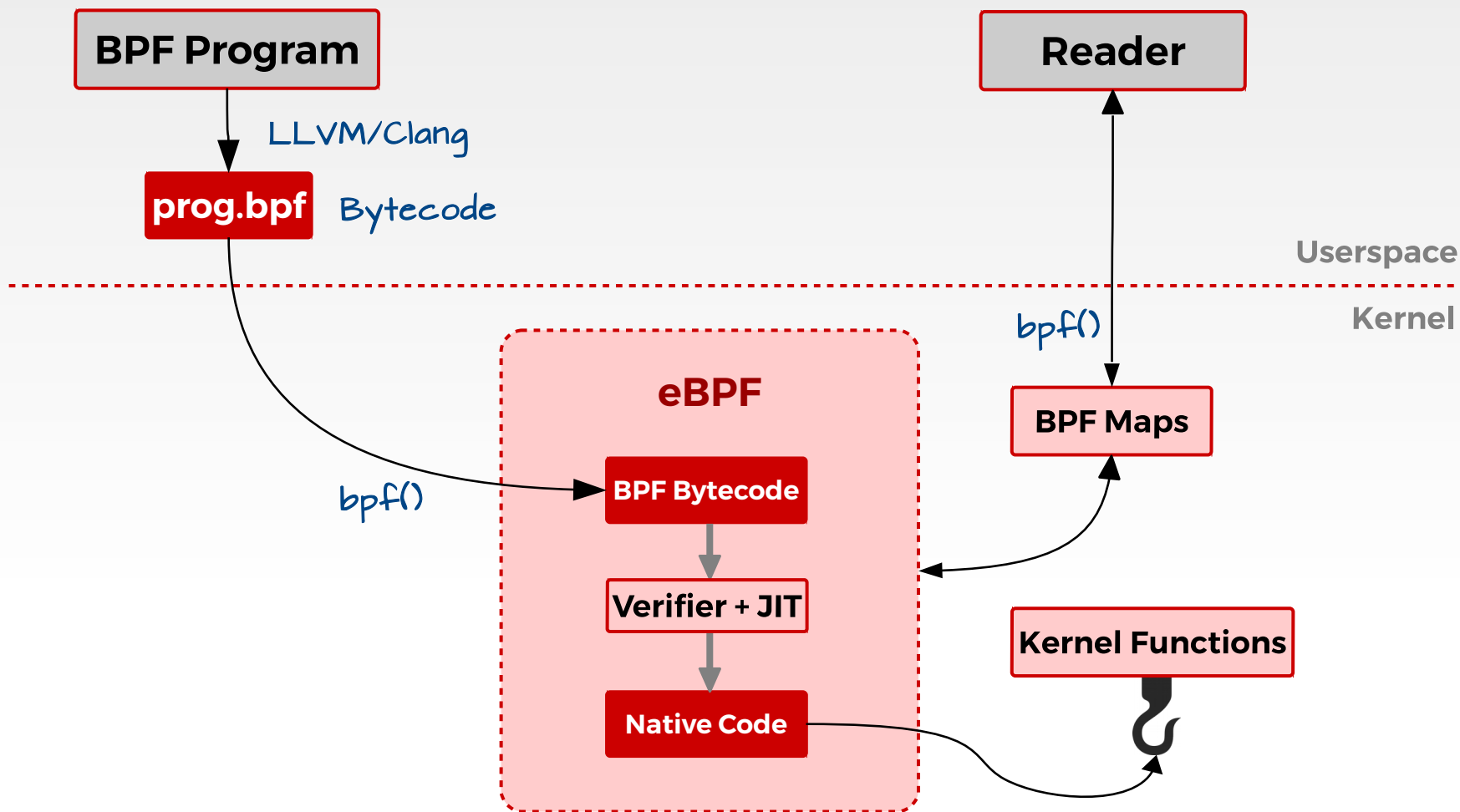
Extended BPF (eBPF) [Starovoitov 2015] [Clément 2016] [Sharma et al. 2016]

- More registers, JIT compiler (flexible/faster), verifier
- Attach on Tracepoint/Kprobe/Uprobe/USDT
- In-kernel trace aggregation & filtering
- Control via ***bpf()***, trace collection via **BPF Maps**
- Upstream in Linux Kernel (***bpf()*** syscall, v3.18+)
- Bytecode compilation upstream in LLVM/Clang

*JIT support eventually landed in kernel

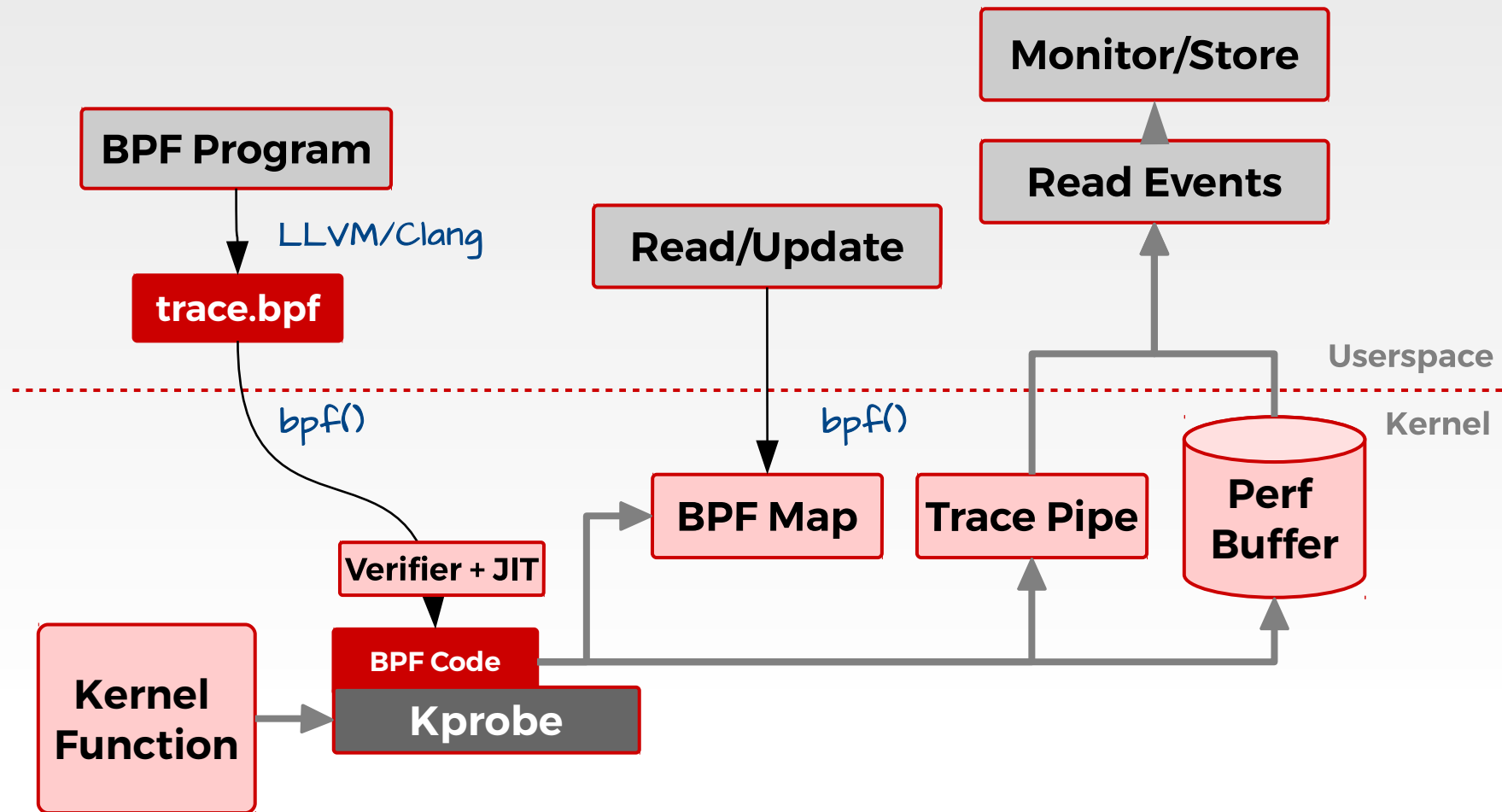
Berkeley Packet Filter

Program Anatomy



eBPF for Tracing

Kprobes/Kretprobes



Clang-Go Demo Time!

eBPF Features & Support

Major BPF Milestones by Kernel Version*

- 3.18 : bpf() syscall
- 3.19 : Sockets support, BPF Maps
- 4.1 : Kprobe support
- 4.4 : Perf events
- 4.6 : Stack traces, per-CPU Maps
- 4.7 : Attach on Tracepoints
- 4.8 : XDP core and act
- 4.9 : Profiling, attach to Perf events
- 4.10 : cgroups support (socket filters)
- 4.11 : *Tracerception* – tracepoints for eBPF debugging

*Adapted from “BPF: Tracing and More” by Brendan Gregg (Linux.Conf.au 2017)

eBPF Features & Support

Program Types

- 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_LANDLOCK**
-
- The diagram shows three categories of program types:
- Tracing** (indicated by blue text and arrows pointing to):
 - BPF_PROG_TYPE_KPROBE
 - BPF_PROG_TYPE_TRACEPOINT
 - BPF_PROG_TYPE_PERF_EVENT
 - Cgroups** (indicated by blue text and arrows pointing to):
 - BPF_PROG_TYPE_CGROUP_SKB
 - BPF_PROG_TYPE_CGROUP_SOCK
 - Security** (indicated by blue text and an arrow pointing to):
 - BPF_PROG_TYPE_LANDLOCK**

eBPF Features & Support

Map Types

- 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

Kernel Code Walkthrough

eBPF Programs

Frontends

- IOVisor BCC – Python, C++, Lua, Go (gobpf) APIs
- Compile BPF programs directly via LLVM interface
- Helper functions to manage maps, buffers, probes

Kprobes Tracing Example

Complete Program
`trace_fields.py`

```
from bcc import BPF
```

```
prog = """  
int hello(void *ctx) {  
    bpf_trace_printk("Hello reserved-bit!\\n");  
    return 0;  
}  
"""
```

} prog compiled to
BPF bytecode

Attach to Kprobe event

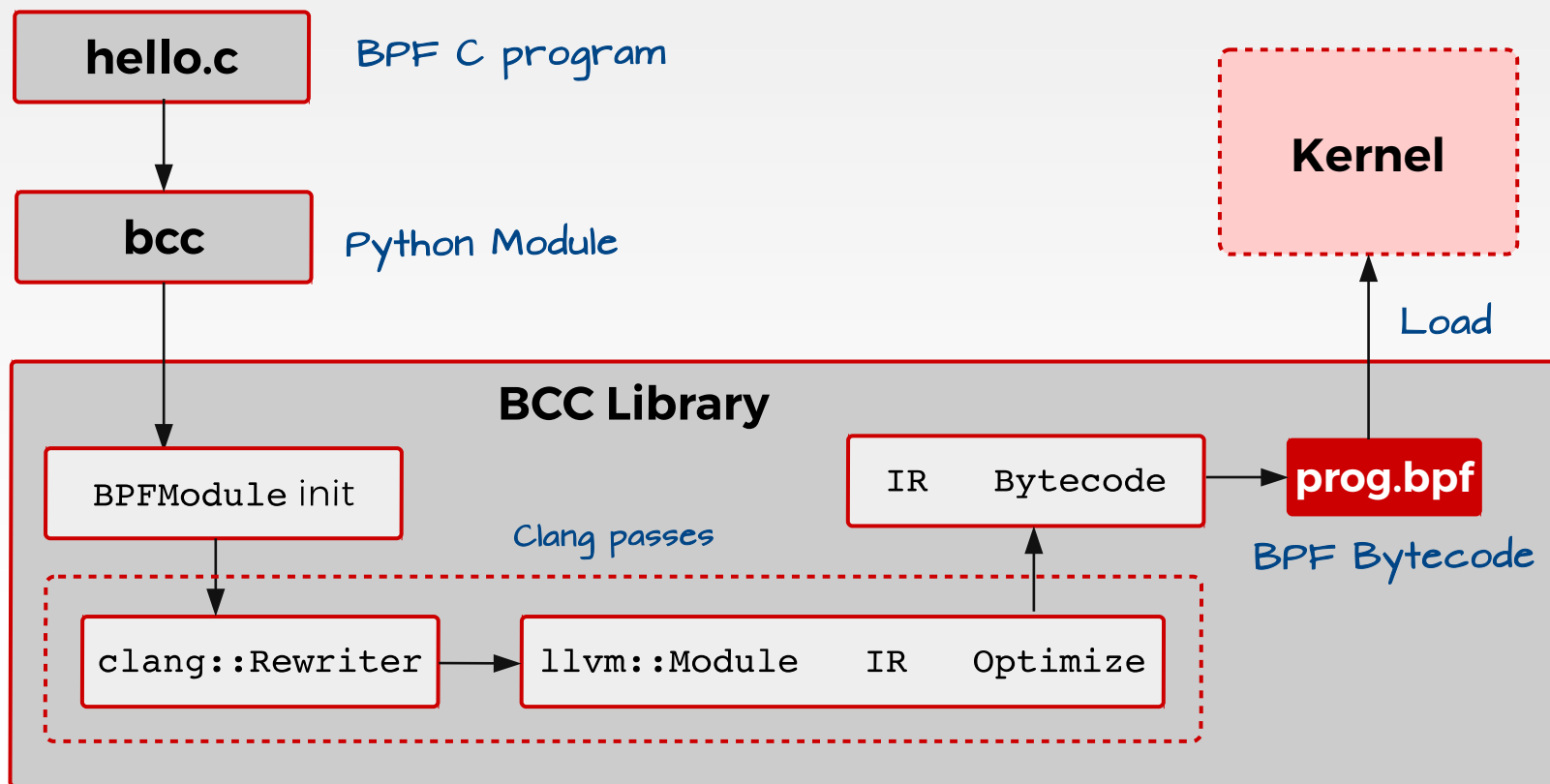
```
b = BPF(text=prog)  
b.attach_kprobe(event="sys_clone", fn_name="hello")  
print "PID MESSAGE"  
b.trace_print(fmt="{1} {5}")
```

Print trace pipe

eBPF Programs

BPF Compiler Collection (BCC)

- Interface to compile, insert and manage BPF code
- libbcc.so is primary interface (list in libbcc.py)



Program Build and Load Example

BCC Demo Time!

eBPF for Tracing

Tracepoint Example (v4.7+)

Program Excerpt

```
# define EXIT_REASON 18

prog = ""
TRACEPOINT_PROBE(kvm, kvm_exit) {
    if (args->exit_reason == EXIT_REASON) {
        bpf_trace_printk("KVM_EXIT exit_reason : %d\\n", args->exit_reason);
    }
    return 0;
}

TRACEPOINT_PROBE(kvm, kvm_entry) {
    if (args->vcpu_id = 0) {
        bpf_trace_printk("KVM_ENTRY vcpu_id : %u\\n", args->vcpu_id);
    }
}
""
```

Attach to tracepoint

Filter on args

Output

```
# ./kvm-test.py
2445.577129000    CPU 0/KVM      8896    KVM_ENTRY vcpu_id : 0
2445.577136000    CPU 0/KVM      8896    KVM_EXIT exit_reason : 18
```

eBPF for Tracing

Uprobes Example

Program Excerpt

```
bpf_text = ""
#include <uapi/linux/ptrace.h>
#include <uapi/linux/limits.h>

int get_fname(struct pt_regs *ctx) {
    if (!ctx->si)
        return 0;
    char str[NAME_MAX] = {};
    bpf_probe_read(&str, sizeof(str), (void *)ctx->si);
    bpf_trace_printk("%s\\n", &str);
    return 0;
};
"""
b = BPF(text=bpf_text)
b.attach_uprobe(name="/usr/bin/vim", sym="readfile", fn_name="get_fname")
```

Get 2nd argument

Process

Symbol

Output

```
# ./vim-test.py
TASK    PID    FILENAME
vim     23707  /tmp/wololo
```

eBPF for Tracing

USDT Example

Program Excerpt
nodejs_http_server.py

```
from bcc import BPF, USDT
.
.
bpf_text = """
#include <uapi/linux/ptrace.h>
int do_trace(struct pt_regs *ctx) {
    uint64_t addr;
    char path[128]={0};
    bpf_usdt_readarg(6, ctx, &addr);
    bpf_probe_read(&path, sizeof(path), (void *)addr);
    bpf_trace_printk("path:%s\\n", path);
    return 0;
};
"""

u = USDT(pid=int(pid))
u.enable_probe(probe="http__server__request", fn_name="do_trace")
b = BPF(text=bpf_text, usdt_contexts=[u])
```

Get 6th Argument

Read to local variable

Target PID

Probe in Node

eBPF for Tracing

USDT Example

Output

```
# ./nodejs_http_server.py 24728
TIME(s)          COMM          PID    ARGS
24653324.561322998 node          24728  path:/index.html
24653335.343401998 node          24728  path:/images/welcome.png
24653340.510164998 node          24728  path:/images/favicon.png
```

Supported Frameworks

- MySQL : --enable-dtrace (Build)
- JVM : -XX:+ExtendedDTraceProbes (Runtime)
- Node : --with-dtrace (Build)
- Python : --with-dtrace (Build)
- Ruby : --enable-dtrace (Build)

eBPF for Tracing

BPF Maps – Filters, States, Counters

Program Excerpt
tcpv4connect.py

```
bpf_text = ""
#include <uapi/linux/ptrace.h>
#include <net/sock.h>
#include <bcc/proto.h>

BPF_HASH(currsock, u32, struct sock *);

int kprobe__tcp_v4_connect(struct pt_regs *ctx, struct sock *sk)
{
    u32 pid = bpf_get_current_pid_tgid();
    // stash the sock ptr for lookup on return
    currsock.update(&pid, &sk);
    return 0;
};
.
.
.
```

Key

Value type

update hash map

eBPF for Tracing

BPF Maps – Filters, States, Counters

Program Excerpt
tcpv4connect.py

```
int kretprobe__tcp_v4_connect(struct pt_regs *ctx)
{
    int ret = PT_REGS_RC(ctx); ← ax reg
    u32 pid = bpf_get_current_pid_tgid(); ← Get Key
    struct sock **skpp;
    skpp = currsock.lookup(&pid); ← Lookup
    if (skpp == 0) {
        return 0; // missed entry
    }
    if (ret != 0) {
        // failed to send SYNC packet, may not have populated
        currsock.delete(&pid); ← Delete
    }

    struct sock *skp = *skpp;
    u32 saddr = 0, daddr = 0;
    u16 dport = 0;
    bpf_probe_read(&saddr, sizeof(saddr), &skp->__sk_common.skc_rcv_saddr);
    bpf_probe_read(&daddr, sizeof(daddr), &skp->__sk_common.skc_daddr);
    bpf_probe_read(&dport, sizeof(dport), &skp->__sk_common.skc_dport);
    bpf_trace_printk("trace_tcp4connect %x %x %d\\n", saddr, daddr, ntohs(dport));
    currsock.delete(&pid); ← Delete
    return 0;
}
"""
```

Read stuff from sock ptr

eBPF for Tracing

BPF Maps – Filters, States, Counters

Output

```
# ./tcpv4connect.py
PID    COMM      SADDR          DADDR          DPORT
1479   telnet    127.0.0.1      127.0.0.1      23
1469   curl      10.201.219.236 54.245.105.25  80
1469   curl      10.201.219.236 54.67.101.145  80
```

More Uses

- Record latency (Δt)
 - biosnoop.py
- Flags for keeping track of events
 - kvm_hypercall.py
- Counting events, histograms
 - cachestat.py
 - cpudist.py

eBPF for Tracing

BPF Perf Event Output

- Build perf events and save to per-cpu perf buffers

```
prog = ""  
#include <linux/sched.h>  
#include <uapi/linux/ptrace.h>  
#include <uapi/linux/limits.h>  
struct data_t {  
    u32 pid;  
    u64 ts;  
    char comm[TASK_COMM_LEN];  
    char fname[NAME_MAX];  
};  
BPF_PERF_OUTPUT(events);  
int handler(struct pt_regs *ctx) {  
    struct data_t data = {};  
    data.pid = bpf_get_current_pid_tgid();  
    data.ts = bpf_ktime_get_ns();  
    bpf_get_current_comm(&data.comm, sizeof(data.comm));  
    bpf_probe_read(&data.fname, sizeof(data.fname),  
                  (void *)PT_REGS_PARM1(ctx));  
    events.perf_submit(ctx, &data, sizeof(data));  
    return 0;  
}  
""
```

Program Excerpt

Event Struct

Init Event

Build Event

Send to buffer

eBPF Trace Visualization

Current State

- Using ASCII histograms, ASCII escape codes
- eBPF trace driven Flamegraphs

Output
argdist.py

```
# ./argdist -H 'p:c:write(int fd, void *buf, size_t len):size_t:len:fd==1'
[01:47:19]
p:c:write(int fd, void *buf, size_t len):size_t:len:fd==1
len : count      distribution
 0 -> 1      : 0      |
 2 -> 3      : 0      |
 4 -> 7      : 0      |
 8 -> 15     : 3      | *****
16 -> 31     : 0      |
32 -> 63     : 5      | *****
64 -> 127    : 13     | *****
```

References

Papers/Presentations

[McCanne et al. 1993] The BSD Packet Filter: A New Architecture for User-level Packet Capture, *Winter USENIX Conference (1993) San Diego*

[Starovoitov 2015] BPF - in-kernel virtual machine, *Collaboration Summit 2015, US*

[Sharma et al. 2016] Enhanced Userspace and In-Kernel Trace Filtering for Production Systems, *J. Comput. Sci. Technol. (2016), Springer US*

[Clément 2016] Linux Kernel packet transmission performance in high-speed networks, *Masters Thesis (2016), KTH, Stockholm*

[Borkmann 2016] Advanced programmability and recent updates with tc's cls_bpf, *NetDev 1.2 (2016) Tokyo*

References

Links

- [IOVisor BPF Docs](#)
- [IOVisor Mailing List](#)
- [bcc Reference Guide](#)
- [bcc Python Developer Tutorial](#)
- [bcc/BPF Blog Posts](#)
- [Dive into BPF: a list of reading material \(Quentin Monnet\)](#)
- [Cilium - Network and Application Security with BPF and XDP \(Thomas Graf\)](#)
- [Landlock LSM Docs \(Mickaël Salaün et al.\)](#)
- [XDP for the Rest of Us \(Jesper Brouer & Andy Gospodarek, Netdev 2.1\)](#)
- [USDT/BPF Tracing Tools \(Sasha Goldshtein\)](#)
- [Linux 4.x Tracing : Performance Analysis with bcc/BPF \(Brendan Gregg, SCALE 15X\)](#)
- [The Common Trace Format \(EfficiOS/Diamon Workgroup\)](#)
- [babeltrace Library \(EfficiOS/Diamon Workgroup\)](#)
- [Trace Compass](#)
- [BPF/bcc for Oracle Tracing](#)
- [Weaveworks Scope HTTP Statistics Plugin](#)

Ack

DORSAL Lab, Polytechnique Montréal

IOVisor Project

Reserved Bit

Fin!

suchakrapani.sharma@polymtl.ca
@tuxology

All the text and images in this presentation drawn by the authors are released under CC-BY-SA. Images not drawn by authors have been attributed either on slides or in references.

