eBPF Powered
Kubernetes Performance Analysis

KubeCon | CloudNativeCon

North America 2018

@fntlnz

eBPF (extended BPF)
Extended Berkeley Packet Filter

@fntlnz

BPF is a Tracing Framework*

Used to access **kernel trace backend** instrumentation tools

*Actually, it’s not just that. And there’s also XDP.

sched:

irq:

signal:

```
# ls /sys/kernel/debug/tracing/events/irq/
enable  filter  irq_handler_entry  irq_handler_exit  softirq_entry  softirq_exit
softirq_raise
```

NEW! tcp:

kvm:

Static tracepoints

```
# cat /sys/kernel/debug/tracing/available_events
syscalls:sys_enter_sendmsg
syscalls:sys_exit_shutdown
syscalls:sys_enter_shutdown
syscalls:sys_exit_getsockopt
syscalls:sys_enter_getsockopt
syscalls:sys_exit_setsockopt
syscalls:sys_enter_setsockopt
syscalls:sys_exit_recvfrom
syscalls:sys_enter_recvfrom
syscalls:sys_exit_sendto
syscalls:sys_enter_sendto
syscalls:sys_exit_getpeername
syscalls:sys_enter_getpeername
syscalls:sys_exit_getsockname
syscalls:sys_enter_getsockname
syscalls:sys_exit_connect
syscalls:sys_enter_connect
syscalls:sys_exit_accept
syscalls:sys_enter_accept
syscalls:sys_exit_accept4
syscalls:sys_enter_accept4
syscalls:sys_exit_listen
syscalls:sys_enter_listen
syscalls:sys_exit_bind
syscalls:sys_enter_bind
syscalls:sys_exit_socketpair
syscalls:sys_enter_socketpair
syscalls:sys_exit_socket
```

timer:

task:

workqueue:

`XDP`

Dynamic ~~trace~~ functionalities

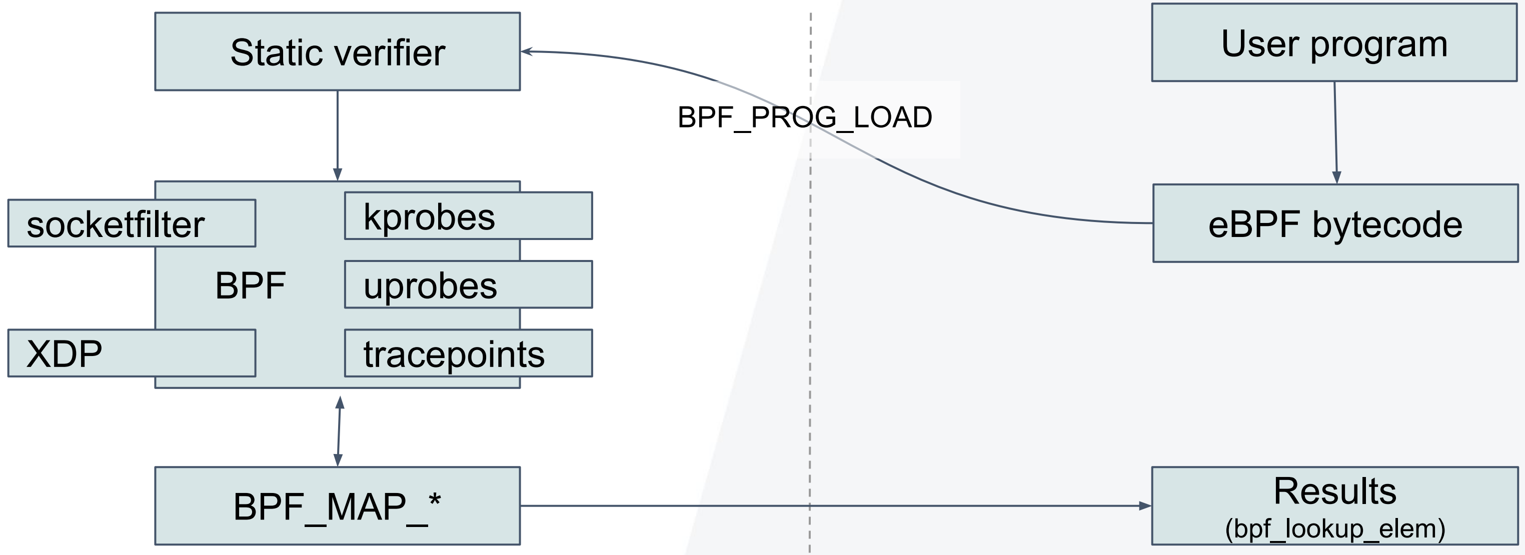
`uprobes`

`kprobes`

Aggregate events at **kernel side** and deal with **just a few** events instead of thousands of them

eBPF program lifecycle

see man 2 bpf



Kernel space

User space

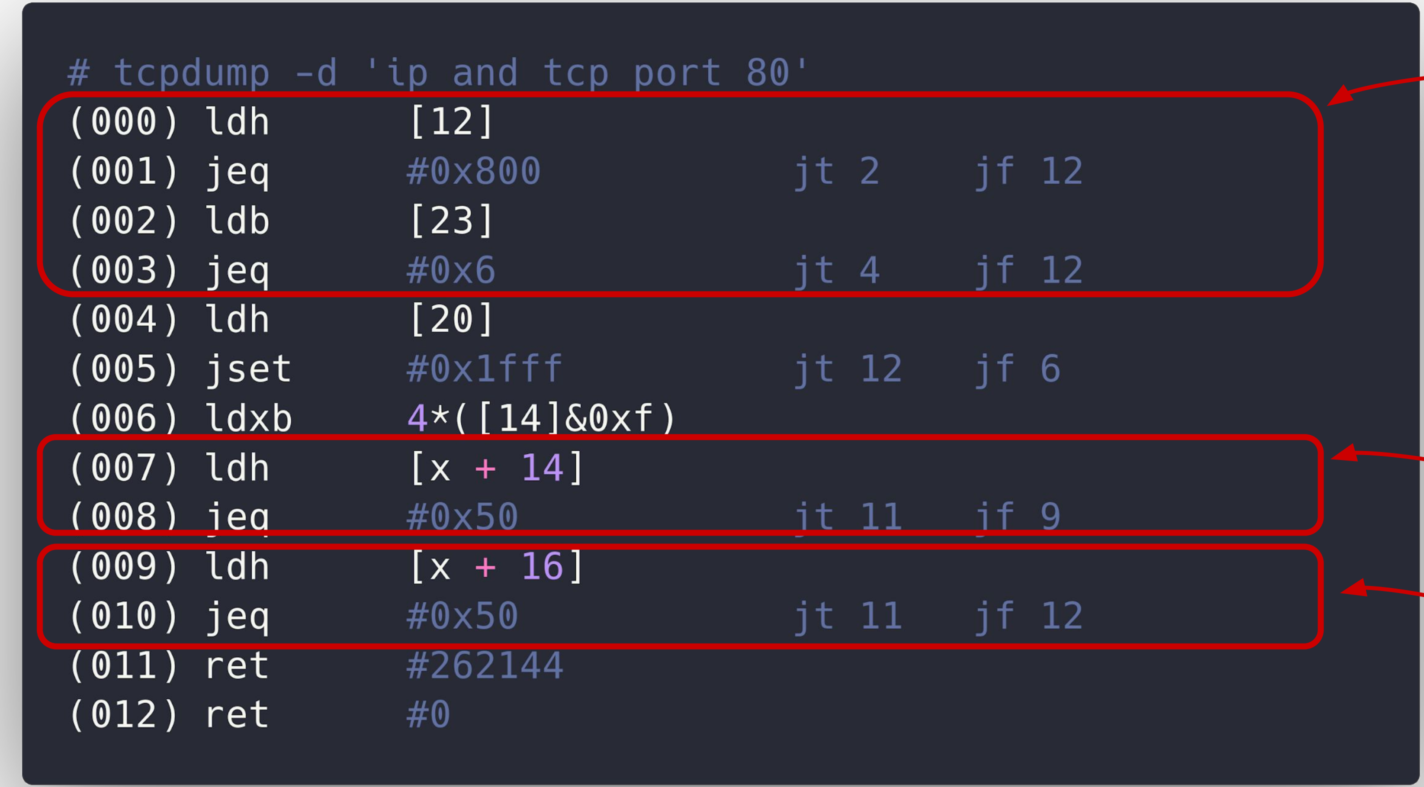
The mustache parrot warns!
eBPF programs can't be turing complete!

In today’s world

@fntlnz

In today’s world: **tcpdump**

`-d` stands for: Dump the compiled packet-matching code in a human readable form to standard output and stop.



```
# tcpdump -d 'ip and tcp port 80'
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 12
(002) ldb      [23]
(003) jeq      #0x6             jt 4    jf 12
(004) ldh      [20]
(005) jset     #0x1fff          jt 12   jf 6
(006) ldxb     4*([14]&0xf)
(007) ldh      [x + 14]
(008) jeq      #0x50            jt 11   jf 9
(009) ldh      [x + 16]
(010) jeq      #0x50            jt 11   jf 12
(011) ret      #262144
(012) ret      #0
```

Is it an ethernet IP IPv4 packet?

Is src (x+14) on port 80 (0x50)?

Is dst (x+16) on port 80 (0x50)?

Documentation about the instruction set: https://www.kernel.org/doc/Documentation/networking/filter.txt

In today's world: **seccomp**

```c
#include <errno.h>
#include <linux/audit.h>
#include <linux/bpf.h>
#include <linux/filter.h>
#include <linux/seccomp.h>
#include <linux/unistd.h>
#include <stddef.h>
#include <stdio.h>
#include <sys/prctl.h>
#include <unistd.h>

static int install_filter(int nr, int arch, int error) {
  struct sock_filter filter[] = {
      BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, arch))),
      BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, arch, 0, 3),
      BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, nr))),
      BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, nr, 0, 1),
      BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ERRNO | (error & SECCOMP_RET_DATA)),
      BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ALLOW),
  };
  struct sock_fprog prog = {
      .len = (unsigned short)(sizeof(filter) / sizeof(filter[0])),
      .filter = filter,
  };
  if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)) {
    perror("prctl(NO_NEW_PRIVS)");
    return 1;
  }
  if (prctl(PR_SET_SECCOMP, 2, &prog)) {
    perror("prctl(PR_SET_SECCOMP)");
    return 1;
  }
  return 0;
}

int main() {
  printf("hey there!\n");

  install_filter(__NR_write, AUDIT_ARCH_X86_64, EPERM);

  printf("something's gonna happen!!\n");
  printf("it will not definitely print this here\n");
  return 0;
}
```

```
gcc main.c
strace ./a.out

...

write(1, "hey there!\n", 11hey there!
)             = 11
prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)  = 0
prctl(PR_SET_SECCOMP, SECCOMP_MODE_FILTER, {len=6, filter=0x7ffe3fd635b0}) = 0
write(1, "something's gonna happen!!\n", 27) = -1 EPERM (Operation not permitted)
write(1, "it will not definitely print thi"..., 39) = -1 EPERM (Operation not
permitted)
exit_group(0)                           = ?
+++ exited with 0 +++
```

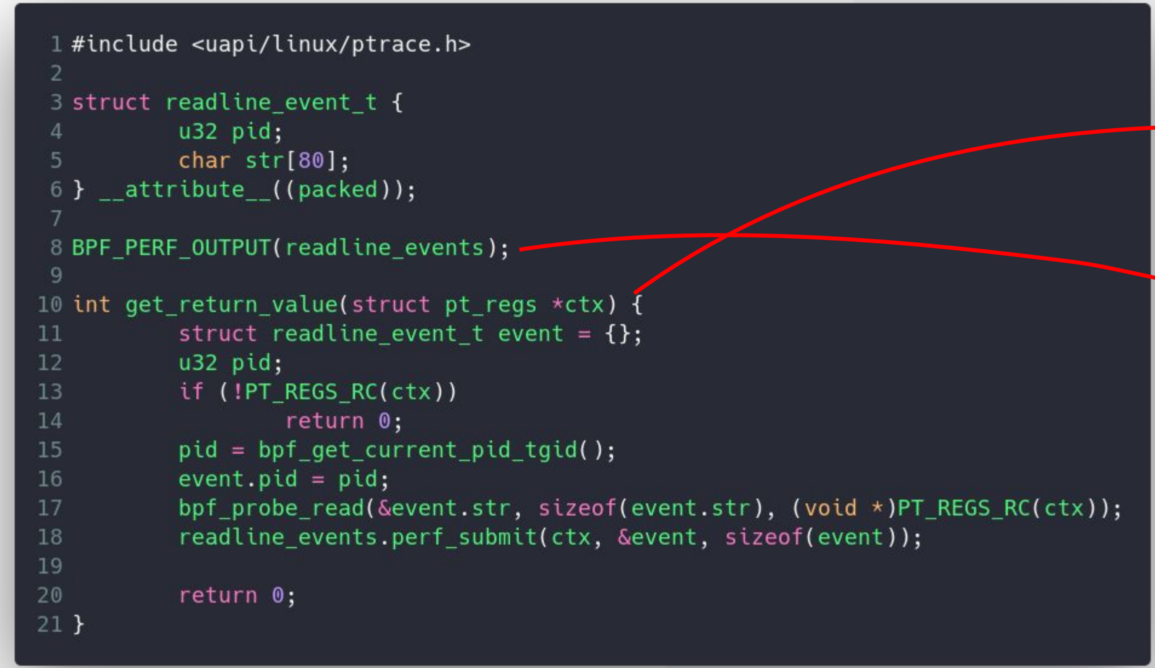
@fntlnz

More practical examples?

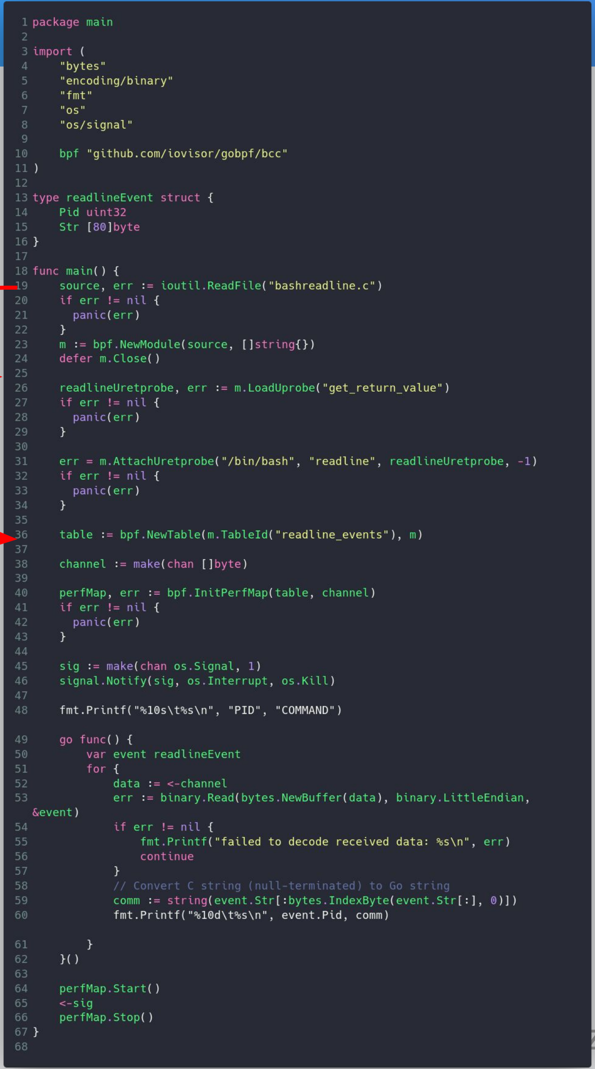
- Trace file opens by filename
- Trace queries done against a database, like InfluxDB or MySQL
- Trace TCP retransmissions
- Trace all commands done in a bash shell
- Trace block device I/O latency over time
- JVM events
- Go Runtime Events
- Firewalls, packet rewriting, dropping etc..

High-level APIs are there!

iovisor/gobpf



```c
#include <uapi/linux/ptrace.h>

struct readline_event_t {
        u32 pid;
        char str[80];
} __attribute__((packed));

BPF_PERF_OUTPUT(readline_events);

int get_return_value(struct pt_regs *ctx) {
        struct readline_event_t event = {};
        u32 pid;
        if (!PT_REGS_RC(ctx))
                return 0;
        pid = bpf_get_current_pid_tgid();
        event.pid = pid;
        bpf_probe_read(&event.str, sizeof(event.str), (void *)PT_REGS_RC(ctx));
        readline_events.perf_submit(ctx, &event, sizeof(event));

        return 0;
}
```



```go
package main

import (
    "bytes"
    "encoding/binary"
    "fmt"
    "os"
    "os/signal"

    bpf "github.com/iovisor/gobpf/bcc"
)

type readlineEvent struct {
    Pid uint32
    Str [80]byte
}

func main() {
    source, err := ioutil.ReadFile("bashreadline.c")
    if err != nil {
      panic(err)
    }
    m := bpf.NewModule(source, []string{})
    defer m.Close()

    readlineUretprobe, err := m.LoadUprobe("get_return_value")
    if err != nil {
      panic(err)
    }

    err = m.AttachUretprobe("/bin/bash", "readline", readlineUretprobe, -1)
    if err != nil {
      panic(err)
    }

    table := bpf.NewTable(m.TableId("readline_events"), m)

    channel := make(chan []byte)

    perfMap, err := bpf.InitPerfMap(table, channel)
    if err != nil {
      panic(err)
    }

    sig := make(chan os.Signal, 1)
    signal.Notify(sig, os.Interrupt, os.Kill)

    fmt.Printf("%10s\t%s\n", "PID", "COMMAND")

    go func() {
        var event readlineEvent
        for {
            data := <-channel
            err := binary.Read(bytes.NewBuffer(data), binary.LittleEndian,
&event)
            if err != nil {
                fmt.Printf("failed to decode received data: %s\n", err)
                continue
            }
            // Convert C string (null-terminated) to Go string
            comm := string(event.Str[:bytes.IndexByte(event.Str[:], 0)])
            fmt.Printf("%10d\t%s\n", event.Pid, comm)

        }
    }()

    perfMap.Start()
    <-sig
    perfMap.Stop()
}
```

iovisor/bpftrace

Inline programs

```
# Read bytes by process:
bpftrace -e 'tracepoint:syscalls:sys_exit_read /args->ret/ { @[comm] =
sum(args->ret); }'

# Read size distribution by process:
bpftrace -e 'tracepoint:syscalls:sys_exit_read { @[comm] = hist(args->ret); }'

# Show per-second syscall rates:
bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @ = count(); } interval:s:1 {
print(@); clear(@); }'
```

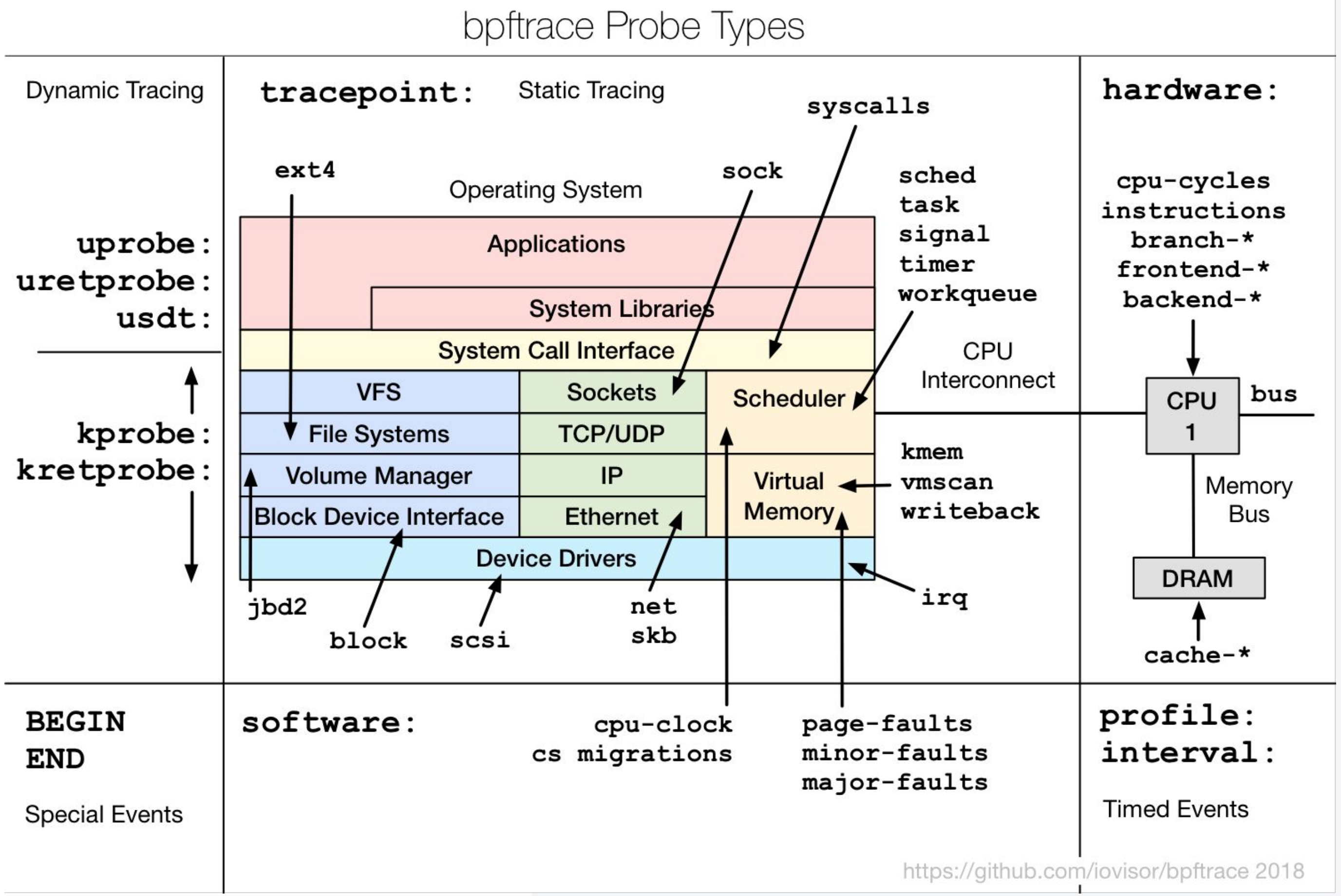
Program from file

```
tracepoint:syscalls:sys_enter_read
{
  @start[tid] = nsecs;
}

tracepoint:syscalls:sys_exit_read / @start[tid] /
{
  @times = hist(nsecs - @start[tid]);
  delete(@start[tid]);
}
```

```
# bpftrace read.bt
Attaching 2 probes...
^C

@times:
[256, 512)           326 |@                                                   |
[512, 1k)           7715 |@@@@@@@@@@@@@@@@@@@@@@@@@@                          |
[1k, 2k)           15306 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[2k, 4k)             609 |@@                                                  |
[4k, 8k)             611 |@@                                                  |
[8k, 16k)            438 |@                                                   |
[16k, 32k)            59 |                                                    |
[32k, 64k)            36 |                                                    |
[64k, 128k)            5 |                                                    |
```

@fntlnz

bpftrace probe types



@fntlnz

What about Kubernetes?

github.com/fntlnz/kubectl-trace

The kubectl trace plugin

Your bpftrace program

```
kubectl trace run -e 'kprobe:do_sys_open { printf("%s,%s\n", comm, str(arg1))
}'  ip-180-12-0-220.ec2.internal -a
```

Attach the terminal to the program's TTY

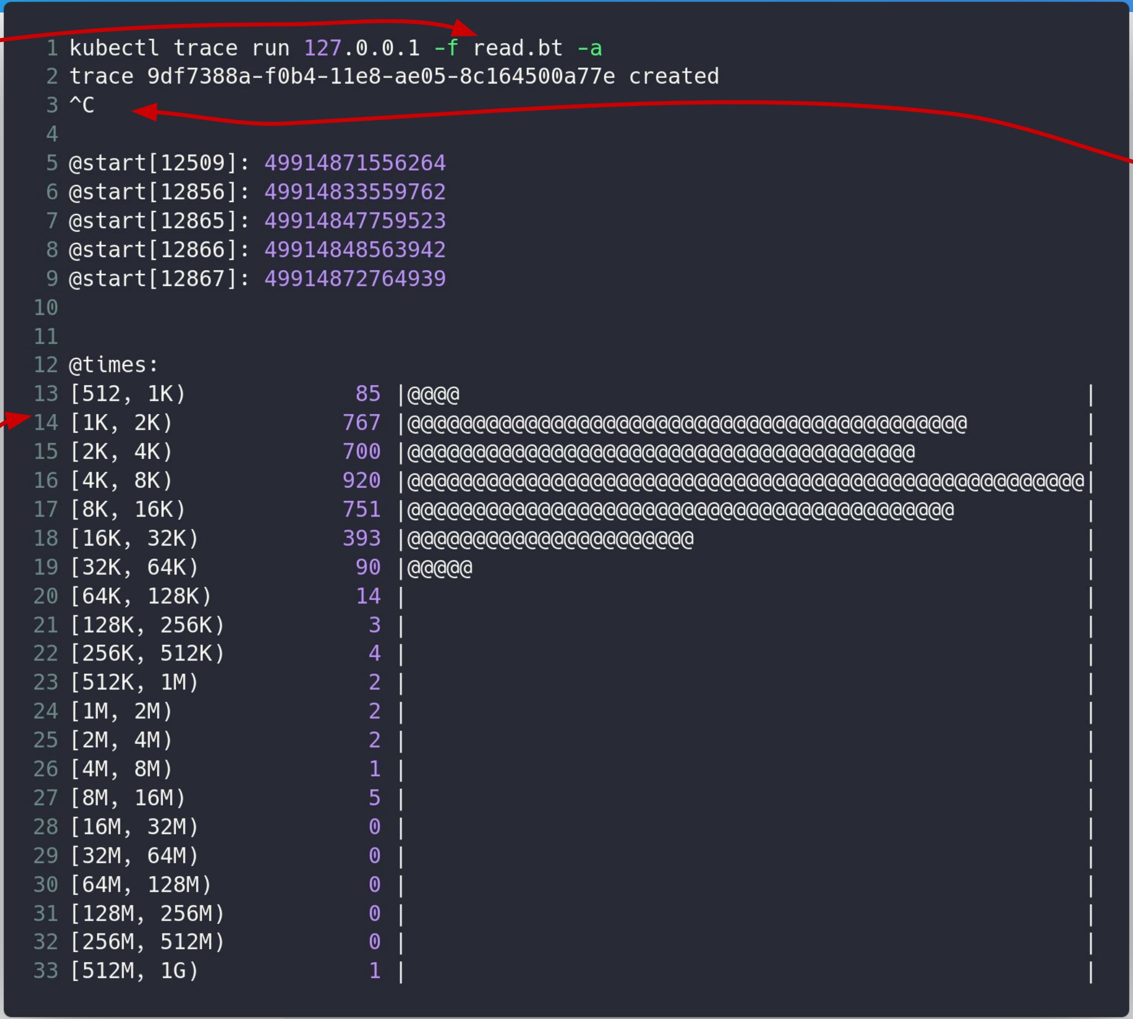
The node where to run it in your cluster

github.com/fntlnz/kubectl-trace

Run program from file

Ctrl-C tells the program to
Plot the results using hist()

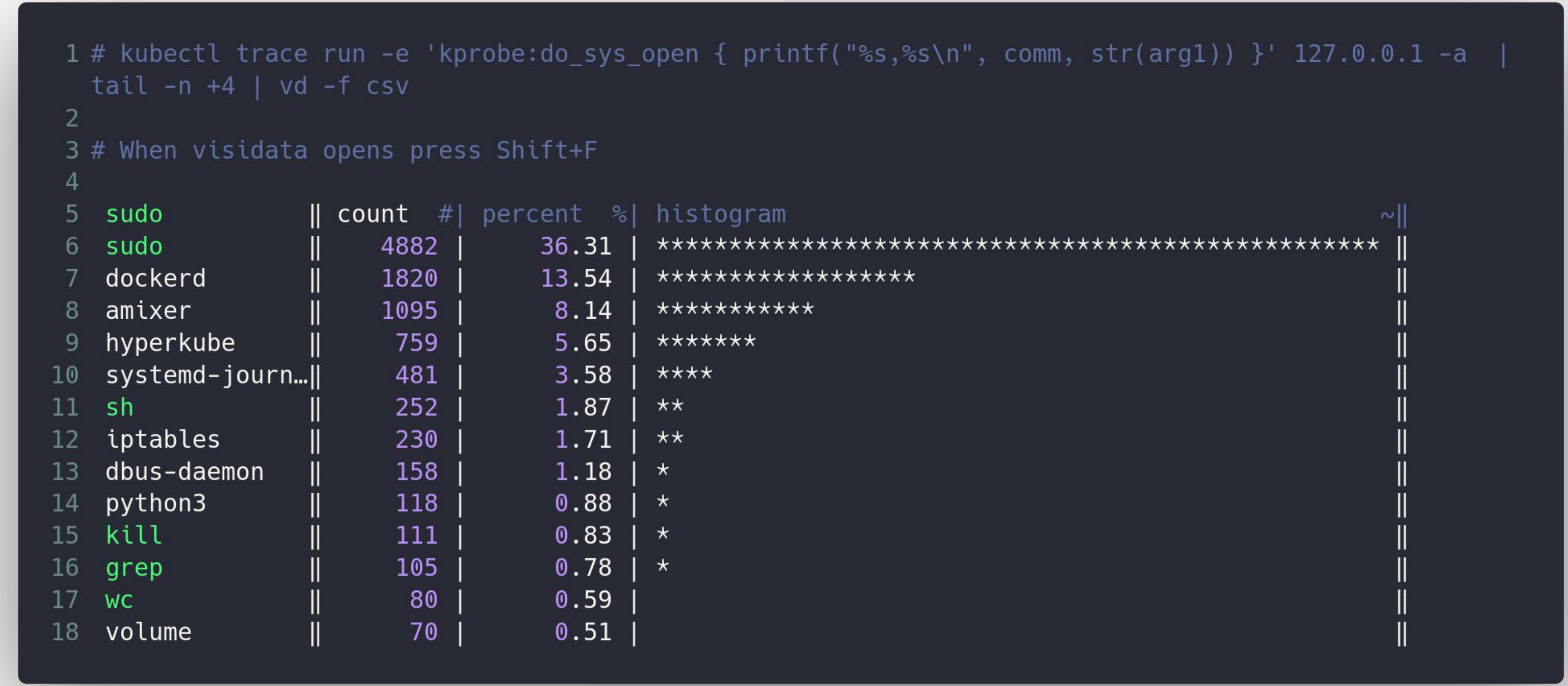
The output histogram



```
kubectl trace run 127.0.0.1 -f read.bt -a
trace 9df7388a-f0b4-11e8-ae05-8c164500a77e created
^C

@start[12509]: 49914871556264
@start[12856]: 49914833559762
@start[12865]: 49914847759523
@start[12866]: 49914848563942
@start[12867]: 49914872764939


@times:
[512, 1K)             85 |@@@@                                                |
[1K, 2K)             767 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@         |
[2K, 4K)             700 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@             |
[4K, 8K)             920 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[8K, 16K)            751 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@          |
[16K, 32K)           393 |@@@@@@@@@@@@@@@@@@@@@@                              |
[32K, 64K)            90 |@@@@@                                               |
[64K, 128K)           14 |                                                    |
[128K, 256K)           3 |                                                    |
[256K, 512K)           4 |                                                    |
[512K, 1M)             2 |                                                    |
[1M, 2M)               2 |                                                    |
[2M, 4M)               2 |                                                    |
[4M, 8M)               1 |                                                    |
[8M, 16M)              5 |                                                    |
[16M, 32M)             0 |                                                    |
[32M, 64M)             0 |                                                    |
[64M, 128M)            0 |                                                    |
[128M, 256M)           0 |                                                    |
[256M, 512M)           0 |                                                    |
[512M, 1G)             1 |                                                    |
```

@fntlnz

kubectl trace and visidata!



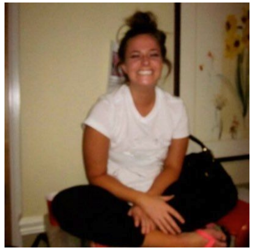
```
# kubectl trace run -e 'kprobe:do_sys_open { printf("%s,%s\n", comm, str(arg1)) }' 127.0.0.1 -a  |
tail -n +4 | vd -f csv

# When visidata opens press Shift+F

 sudo            ‖ count   #| percent  %| histogram                                              ~‖
 sudo            ‖    4882 |      36.31 | ***************************************************  ‖
 dockerd         ‖    1820 |      13.54 | ******************                                   ‖
 amixer          ‖    1095 |       8.14 | ***********                                          ‖
 hyperkube       ‖     759 |       5.65 | *******                                              ‖
 systemd-journ…‖     481 |       3.58 | ****                                                 ‖
 sh              ‖     252 |       1.87 | **                                                   ‖
 iptables        ‖     230 |       1.71 | **                                                   ‖
 dbus-daemon     ‖     158 |       1.18 | *                                                    ‖
 python3         ‖     118 |       0.88 | *                                                    ‖
 kill            ‖     111 |       0.83 | *                                                    ‖
 grep            ‖     105 |       0.78 | *                                                    ‖
 wc              ‖      80 |       0.59 |                                                      ‖
 volume          ‖      70 |       0.51 |                                                      ‖
```

@fntlnz

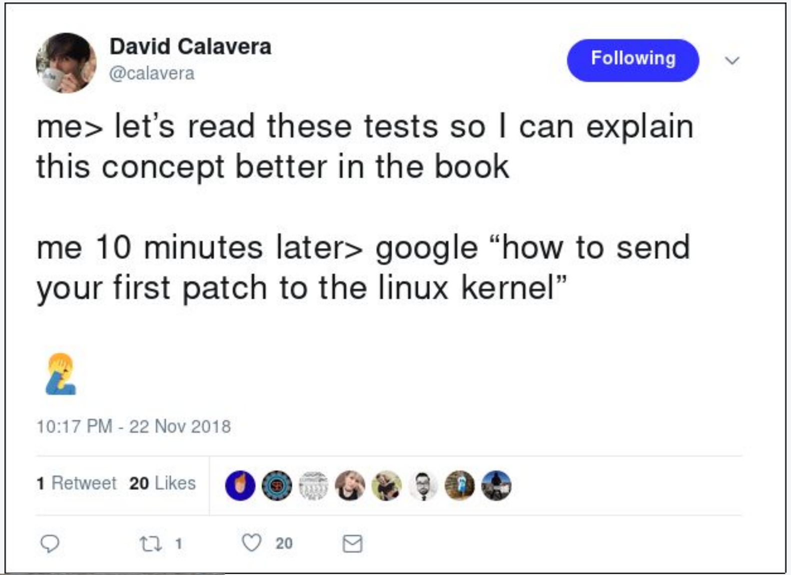
DEMO TIME

@fntlnz

Any BPF Books to recommend ?



David and Jessie are writing one!!



David Calavera
@calavera
Following

me> let's read these tests so I can explain this concept better in the book

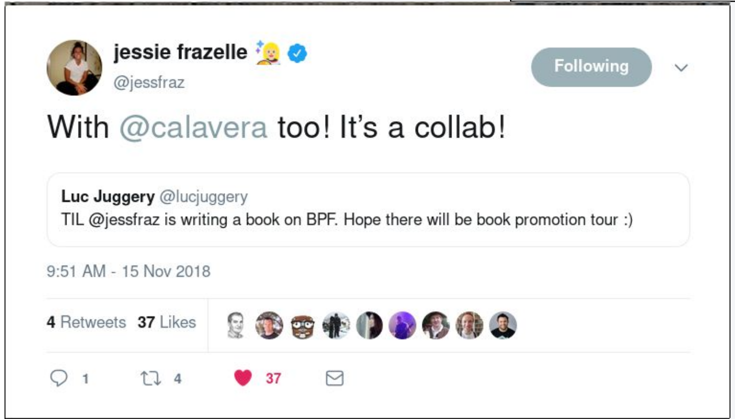
me 10 minutes later> google "how to send your first patch to the linux kernel"

🤦

10:17 PM - 22 Nov 2018

1 Retweet 20 Likes

1 20



jessie frazelle 🧚
@jessfraz
Following

With @calavera too! It's a collab!

Luc Juggery @lucjuggery
TIL @jessfraz is writing a book on BPF. Hope there will be book promotion tour :)

9:51 AM - 15 Nov 2018

4 Retweets 37 Likes

1 4 37

KubeCon | CloudNativeCon
North America 2018

References

1. iovisor BCC
2. Cilium: HTTP, gRPC, and Kafka Aware Security and Networking for Containers with BPF and XDP
3. iovisor/gobpf - To load eBPF programs using Go
4. Landlock LSM
5. iovisor bpftrace
6. iovisor BPF docs
7. Blog post on how to load xdp programs using iproute2
8. BPF Tracing Talk from Brendan Gregg
9. Cilium documentation for BPF

1. https://www.iovisor.org/
2. https://github.com/cilium/cilium
3. https://github.com/iovisor/gobpf
4. https://landlock.io/
5. https://github.com/iovisor/bpftrace
6. https://github.com/iovisor/bpf-docs
7. https://medium.com/@fntlnz/load-xdp-programs-using-the-ip-iproute2-command-502043898263
8. https://www.youtube.com/watch?v=JRFNIKUROPE
9. https://cilium.readthedocs.io/en/latest/bpf/

KubeCon | CloudNativeCon

North America 2018

Thank you!

lorenzo@influxdata.com
twitter.com/fntlnz
github.com/fntlnz

If you're that kind of person can find my pgp key here:
https://fntlnz.wtf/downloads/pubkey-B2400EE4.asc

@fntlnz