

eBPF Powered Kubernetes Performance Analysis

The Kubernetes logo, a stylized white cube with internal lines forming a complex geometric pattern, is positioned in the upper right area of the slide. It is set against a blue background that features a subtle, larger-scale geometric pattern of interconnected lines.

FOSDEM 2019

Lorenzo Fontana
@fntlnz

@fntlnz



eBPF (extended BPF) Extended Berkeley Packet Filter

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:



tcp:

kvm:

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

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_getsockopt  
syscalls:sys_enter_setsockopt  
syscalls:sys_exit_rcvfrom  
syscalls:sys_enter_rcvfrom  
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:

workqueue:

task:

XDP

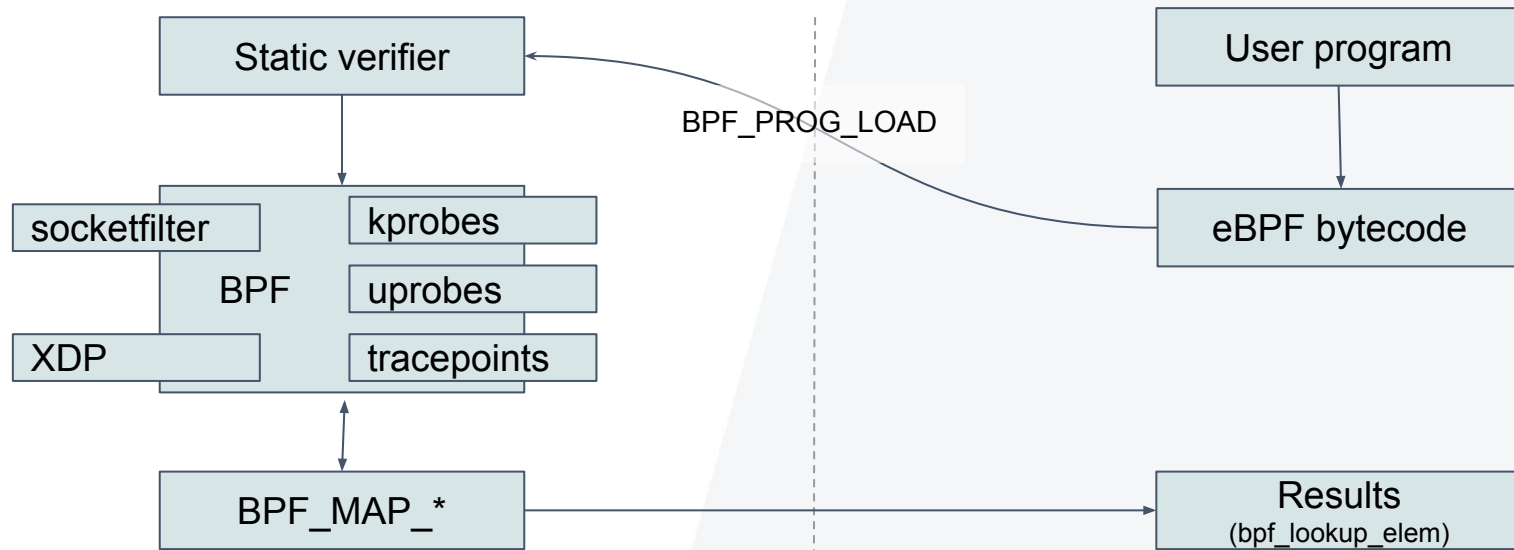
Dynamic ~~trace~~ functionalities

uprobes

kprobes

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

see `man 2 bpf`



Kernel space

User space

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





In today's world

-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) ldx      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)?

In today's world: seccomp

```
1 #include <errno.h>
2 #include <linux/audit.h>
3 #include <linux/bpf.h>
4 #include <linux/filter.h>
5 #include <linux/seccomp.h>
6 #include <linux/unistd.h>
7 #include <stddef.h>
8 #include <stdio.h>
9 #include <sys/prctl.h>
10 #include <unistd.h>
11
12 static int install_filter(int nr, int arch, int error) {
13     struct sock_filter filter[] = {
14         BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, arch))),
15         BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, arch, 0, 3),
16         BPF_STMT(BPF_LD + BPF_W + BPF_ABS, (offsetof(struct seccomp_data, nr))),
17         BPF_JUMP(BPF_JMP + BPF_JEQ + BPF_K, nr, 0, 1),
18         BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ERRNO | (error & SECCOMP_RET_DATA)),
19         BPF_STMT(BPF_RET + BPF_K, SECCOMP_RET_ALLOW),
20     };
21     struct sock_fprog prog = {
22         .len = (unsigned short)(sizeof(filter) / sizeof(filter[0])),
23         .filter = filter,
24     };
25     if (prctl(PR_SET_NO_NEW_PRIVS, 1, 0, 0, 0)) {
26         perror("prctl(NO_NEW_PRIVS)");
27         return 1;
28     }
29     if (prctl(PR_SET_SECCOMP, 2, &prog)) {
30         perror("prctl(PR_SET_SECCOMP)");
31         return 1;
32     }
33     return 0;
34 }
35
36 int main() {
37     printf("hey there!\n");
38
39     install_filter(__NR_write, AUDIT_ARCH_X86_64, EPERM);
40
41     printf("something's gonna happen!!\n");
42     printf("it will not definitely print this here\n");
43     return 0;
44 }
45
```

```
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 +++

= ?
```

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!

```

1 #include <uapi/linux/ptrace.h>
2
3 struct readline_event_t {
4     u32 pid;
5     char str[80];
6 } __attribute__((packed));
7
8 BPF_PERF_OUTPUT(readline_events);
9
10 int get_return_value(struct pt_regs *ctx) {
11     struct readline_event_t event = {};
12     u32 pid;
13     if (!PT_REGS_RC(ctx))
14         return 0;
15     pid = bpf_get_current_pid_tgid();
16     event.pid = pid;
17     bpf_probe_read(&event.str, sizeof(event.str), (void *)PT_REGS_RC(ctx));
18     readline_events.perf_submit(ctx, &event, sizeof(event));
19
20     return 0;
21 }

```

```

1 package main
2
3 import (
4     "bytes"
5     "encoding/binary"
6     "fmt"
7     "os"
8     "os/signal"
9
10     bpf "github.com/iovisor/gobpf/bcc"
11 )
12
13 type readlineEvent struct {
14     Pid uint32
15     Str [80]byte
16 }
17
18 func main() {
19     source, err := ioutil.ReadFile("bashreadline.c")
20     if err != nil {
21         panic(err)
22     }
23     m := bpf.NewModule(source, []string{})
24     defer m.Close()
25
26     readlineUretprobe, err := m.LoadUprobe("get_return_value")
27     if err != nil {
28         panic(err)
29     }
30
31     err = m.AttachUretprobe("/bin/bash", "readline", readlineUretprobe, -1)
32     if err != nil {
33         panic(err)
34     }
35
36     table := bpf.NewTable(m.TableId("readline_events"), m)
37
38     channel := make(chan []byte)
39
40     perfMap, err := bpf.InitPerfMap(table, channel)
41     if err != nil {
42         panic(err)
43     }
44
45     sig := make(chan os.Signal, 1)
46     signal.Notify(sig, os.Interrupt, os.Kill)
47
48     fmt.Printf("%10s\t%s\n", "PID", "COMMAND")
49
50     go func() {
51         var event readlineEvent
52         for {
53             data := <-channel
54             err := binary.Read(bytes.NewBuffer(data), binary.LittleEndian,
55                 &event)
56             if err != nil {
57                 fmt.Printf("failed to decode received data: %s\n", err)
58                 continue
59             }
60             // Convert C string (null-terminated) to Go string
61             comm := string(event.Str[:bytes.IndexByte(event.Str[:], 0)])
62             fmt.Printf("%10d\t%s\n", event.Pid, comm)
63         }
64     }()
65
66     perfMap.Start()
67     <-sig
68     perfMap.Stop()
69 }

```


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(@); }'
```

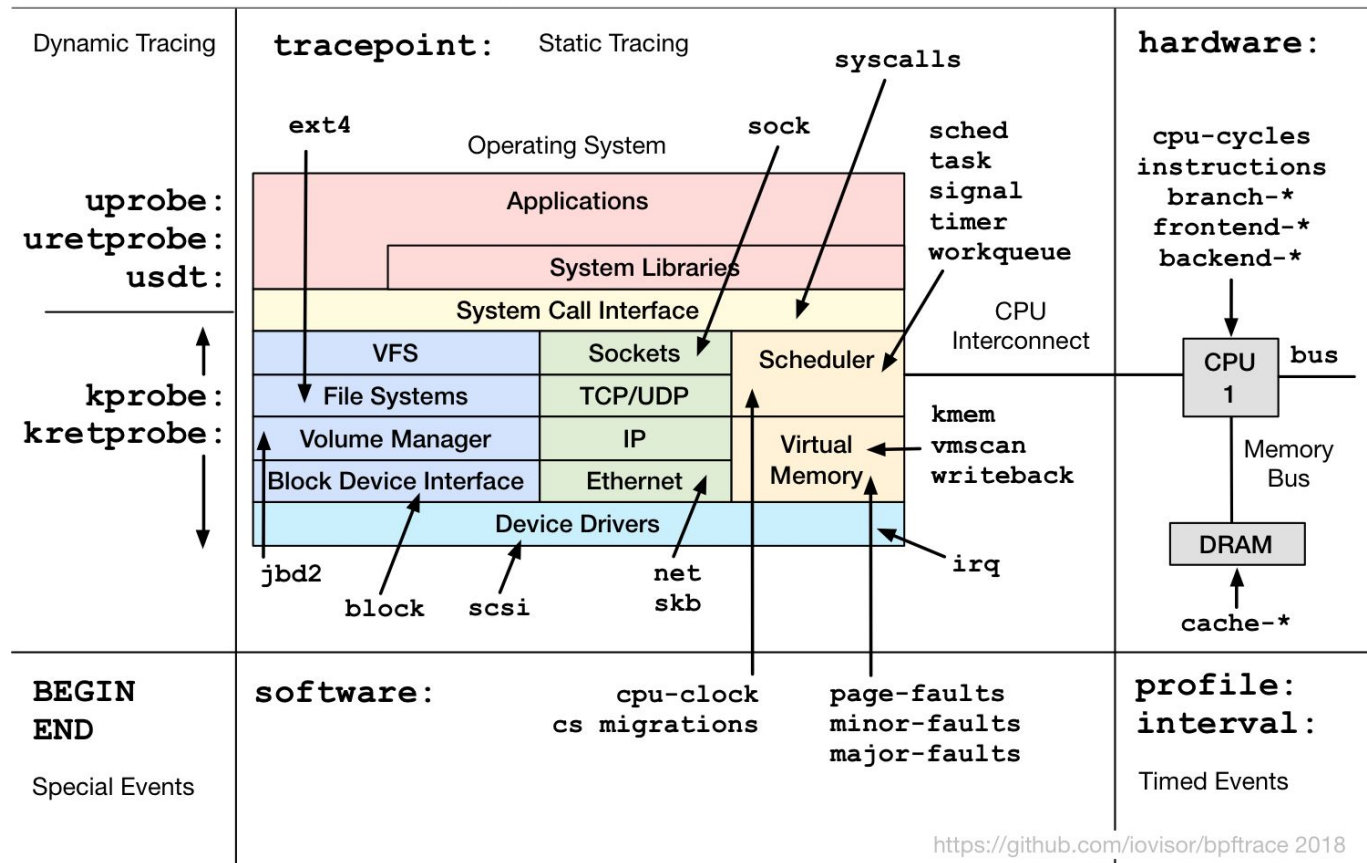
Program from file

```
1 tracepoint:syscalls:sys_enter_read
2 {
3   @start[tid] = nsecs;
4 }
5
6 tracepoint:syscalls:sys_exit_read / @start[tid] /
7 {
8   @times = hist(nsecs - @start[tid]);
9   delete(@start[tid]);
10 }
```

```
# 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  |
```

bpftrace Probe Types



<https://github.com/iovisor/bpfttrace> 2018



What about Kubernetes?



Brendan Gregg

@brendangregg



Welcome kubectl-trace by [@fntlnz](#) to [@iovisor](#), the Linux Foundation eBPF project that hosts bcc and bpftool. kubectl-trace runs bpftool on Kubernetes.



iovisor/kubectl-trace

Schedule bpftool programs on your kubernetes cluster using the kubectl - iovisor/kubectl-trace

[github.com](#)

10:56 PM · Jan 3, 2019 · TweetDeck

117 Retweets

306 Likes



@fntlnz

The kubectl trace plugin

Your bpftrace program

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

The node where to run it in your cluster

Attach the terminal to the program's TTY

Run program from file

```
1 kubectrl trace run 127.0.0.1 -f read.bt -a
2 trace 9df7388a-f0b4-11e8-ae05-8c164500a77e created
3 ^C
4
5 @start[12509]: 49914871556264
6 @start[12856]: 49914833559762
7 @start[12865]: 49914847759523
8 @start[12866]: 49914848563942
9 @start[12867]: 49914872764939
```

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

```
10
11
12 @times:
13 [512, 1K)      85 | @@@@
14 [1K, 2K)      767 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
15 [2K, 4K)      700 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
16 [4K, 8K)      920 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
17 [8K, 16K)     751 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
18 [16K, 32K)    393 | @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
19 [32K, 64K)     90 | @@@@
20 [64K, 128K)   14 |
21 [128K, 256K)  3 |
22 [256K, 512K)  4 |
23 [512K, 1M)    2 |
24 [1M, 2M)      2 |
25 [2M, 4M)      2 |
26 [4M, 8M)      1 |
27 [8M, 16M)     5 |
28 [16M, 32M)    0 |
29 [32M, 64M)    0 |
30 [64M, 128M)   0 |
31 [128M, 256M)  0 |
32 [256M, 512M)  0 |
33 [512M, 1G)    1 |
```

The output histogram

```

1 # 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
2
3 # When visidata opens press Shift+F
4
5 sudo          || count  #| percent %| histogram                                     ~||
6 sudo          || 4882 | 36.31 | *****|
7 dockerd       || 1820 | 13.54 | *****|
8 amixer        || 1095 | 8.14  | *****|
9 hyperkube     || 759  | 5.65  | *****|
10 systemd-journ...|| 481  | 3.58  | ****|
11 sh            || 252  | 1.87  | **|
12 iptables      || 230  | 1.71  | **|
13 dbus-daemon   || 158  | 1.18  | *|
14 python3       || 118  | 0.88  | *|
15 kill          || 111  | 0.83  | *|
16 grep          || 105  | 0.78  | *|
17 wc            || 80   | 0.59  |
18 volume        || 70   | 0.51  |

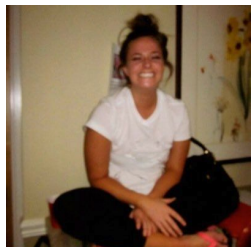
```

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



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=JRFNlKUROPE>
9. <https://cilium.readthedocs.io/en/latest/bpf/>



Thank you!

lorenzo@influxdata.com
lo@bpf.sh

[**twitter.com/fntlnz**](https://twitter.com/fntlnz)
[**github.com/fntlnz**](https://github.com/fntlnz)

If you're that kind of person can find my pgp key here:

@fntlnz