



# Linux超能力BPF技术 学习分享

大卫李 @nevermosby

## 背景介绍

---

近两年BPF技术跃然成为了一项热门技术，在刚刚结束的Kubecon 2020 Europe会议上，有7个与BPF相关的技术分享。

而Kubecon 2020 China会议上也已有了3个关于BPF技术的中文分享，分别来自腾讯和PingCAP，涉足网络优化和系统追踪等领域。在中文社区里，包括阿里巴巴、网易、字节跳动等国内第一梯队IT公司也越来越关注BPF这项新技术。

本次分享主要介绍了BPF技术发展和应用，以及我是如何学习BPF技术的。

# 内容大纲

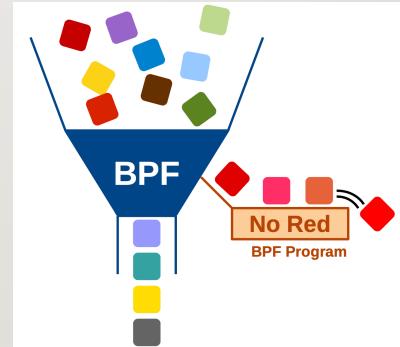
---

- BPF是什么
- BPF技术发展
- BPF技术的超能力是什么
- BPF应用场景
- BPF社区和生态
- 我是如何学习BPF技术的

# BPF是什么

---

- 全称：Berkeley Packet Filter，伯克利包过滤器
- 发明之初是一款网络过滤神器，`tcpdump`就是基于这款神器的神器
- 发展到现在名称升级为eBPF：extended Berkeley Packet Filter
  - 演进成一套通用执行引擎，不再仅仅是网络分析，可以基于eBPF开发性能分析、系统追踪、网络优化等多种类型的工具和平台。
  - 原来的 BPF 就被称为cBPF（classic BPF），目前已基本废弃。当前Linux 内核只运行 eBPF，内核会将cBPF 透明转换成 eBPF 再执行。
- 下文提到的BPF字样没有特别说明的话，是泛指cBPF和eBPF。



# BPF技术发展 — 时间轴

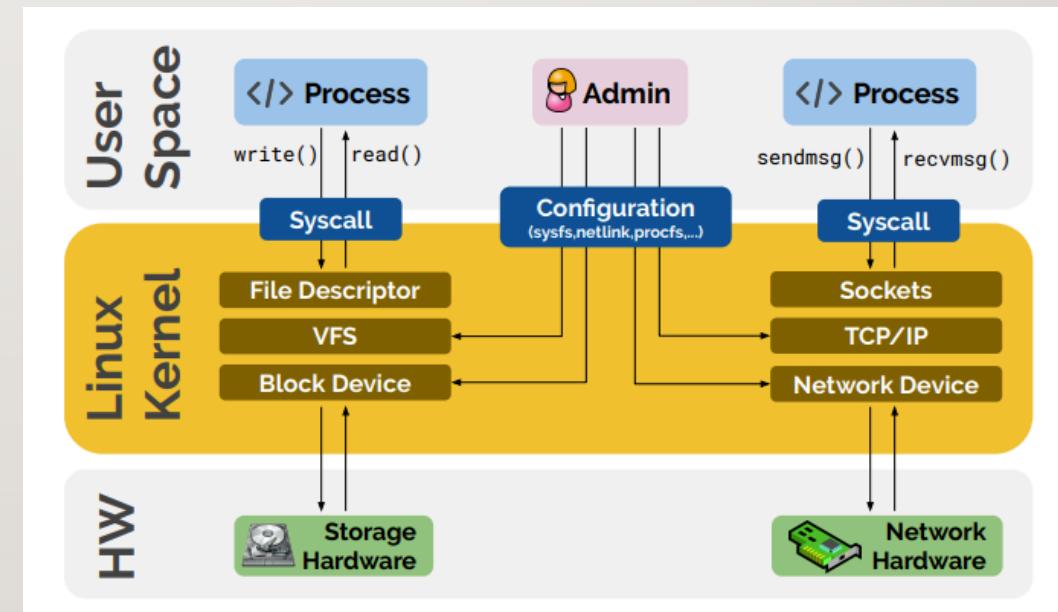


# BPF超能力-解决什么困难

## Linux内核开发

- 强安全
- 高性能
- 持续交付

要实现以上的目标，没有想象中那么简单 ~



# BPF超能力-内核开发一般解决方案和缺陷

---

- 直接修改内核代码，通过API暴露能力，可能要等上n年用户才能更新到这个版本来使用；
- 开发新的可即时加载的内核模块，这个内核模块可能会随着每一个内核版本的发布而不可用，你得非常小心地为每次的内核版本更新调整模块代码，不然就会让内核直接崩溃；

# BPF超能力- BPF解决方案

---

- 强安全：BPF验证器（verifier）通过沙盒机制检查每个程序是否能够安全运行在内核中，拒绝不安全的代码
- 高性能：JIT编译保证了程序本地运行的高性能
- 持续交付：程序可以被无缝替换，并且不影响正在运行的任务。

*“Super powers have finally come to Linux” — Brendan Gregg*

# BPF超能力- 技能点- BPF钩子 (HOOKS)

- BPF 钩子 (Hooks) , 能够加载BPF程序的地方
  - kernel functions (kprobes)
  - userspace functions (uprobes)
  - system calls
  - fentry/fexit
  - Tracepoints
  - network devices (tc/xdp)
  - network routes
  - TCP congestion algorithms
  - sockets (data level)

```
76 static int load_and_attach(const char *event, struct bpf_insn *prog, int size)
77 {
78     bool is_socket = strncmp(event, "socket", 6) == 0;
79     bool is_kprobe = strncmp(event, "kprobe/", 7) == 0;
80     bool is_kretprobe = strncmp(event, "kretprobe/", 10) == 0;
81     bool is_tracepoint = strncmp(event, "tracepoint/", 11) == 0;
82     bool is_raw_tracepoint = strncmp(event, "raw_tracepoint/", 15) == 0;
83     bool is_xdp = strncmp(event, "xdp", 3) == 0;
84     bool is_perf_event = strncmp(event, "perf_event", 10) == 0;
85     bool is_cgroup_skb = strncmp(event, "cgroup(skb", 10) == 0;
86     bool is_cgroup_sk = strncmp(event, "cgroup(sock", 11) == 0;
87     bool is_sockops = strncmp(event, "sockops", 7) == 0;
88     bool is_sk_skb = strncmp(event, "sk_skb", 6) == 0;
89     bool is_sk_msg = strncmp(event, "sk_msg", 6) == 0;
90     size_t insns_cnt = size / sizeof(struct bpf_insn);
91     enum bpf_prog_type prog_type;
92     char buf[256];
93     int fd, efd, err, id;
94     struct perf_event_attr attr = {};
95
96     attr.type = PERF_TYPE_TRACEPOINT;
97     attr.sample_type = PERF_SAMPLE_RAW;
98     attr.sample_period = 1;
99     attr.wakeup_events = 1;
100
101    if (is_socket) {
102        prog_type = BPF_PROG_TYPE_SOCKET_FILTER;
103    } else if (is_kprobe || is_kretprobe) {
104        prog_type = BPF_PROG_TYPE_KPROBE;
105    } else if (is_tracepoint) {
106        prog_type = BPF_PROG_TYPE_TRACEPOINT;
107    } else if (is_raw_tracepoint) {
108        prog_type = BPF_PROG_TYPE_RAW_TRACEPOINT;
109    } else if (is_xdp) {
110        prog_type = BPF_PROG_TYPE_XDP;
111    } else if (is_perf_event) {
112        prog_type = BPF_PROG_TYPE_PERF_EVENT;
113    } else if (is_cgroup_skb) {
114        prog_type = BPF_PROG_TYPE_CGROUP_SKB;
115    } else if (is_cgroup_sk) {
```

linux v5.8.7

# BPF超能力 - 技能点 - BPF MAP

- 
- BPF Map, 存储数据, 交换信息的桥梁
  - Map类型
    - Hash tables, Arrays
    - LRU (Least Recently Used)
    - Ring Buffer
    - Stack Trace
    - LPM (Longest Prefix match)
  - Map功能
    - Program state
    - Program configuration
    - Share data between programs
    - Share state, metrics, and statistics with user space

```
1. struct bpf_map_def SEC("maps") my_bpf_map = {  
2.     .type      = BPF_MAP_TYPE_HASH,  
3.     .key_size   = sizeof(int),  
4.     .value_size = sizeof(int),  
5.     .max_entries = 100,  
6.     .map_flags  = BPF_F_NO_PREALLOC,  
7. };
```

# BPF超能力 - 技能点 - BPF HELPERS

- 辅助函数清单

- Random numbers
- Get current time
- Map access
- Get process/cgroup context
- Manipulate network packets and forwarding
- Access socket data
- Perform tail call
- Access process stack
- Access syscall arguments
- . . .

```
17 /* helper functions called from eBPF programs written in C */
18 static void *(*bpf_map_lookup_elem)(void *map, void *key) =
19         (void *) BPF_FUNC_map_lookup_elem;
20 static int (*bpf_map_update_elem)(void *map, void *key, void *value,
21                                 unsigned long long flags) =
22         (void *) BPF_FUNC_map_update_elem;
23 static int (*bpf_map_delete_elem)(void *map, void *key) =
24         (void *) BPF_FUNC_map_delete_elem;
25 static int (*bpf_probe_read)(void *dst, int size, void *unsafe_ptr) =
26         (void *) BPF_FUNC_probe_read;
27 static unsigned long long (*bpf_ktime_get_ns)(void) =
28         (void *) BPF_FUNC_ktime_get_ns;
29 static int (*bpf_trace_printk)(const char *fmt, int fmt_size, ...) =
30         (void *) BPF_FUNC_trace_printk;
31 static void (*bpf_tail_call)(void *ctx, void *map, int index) =
32         (void *) BPF_FUNC_tail_call;
33 static unsigned long long (*bpf_get_smp_processor_id)(void) =
34         (void *) BPF_FUNC_get_smp_processor_id;
35 static unsigned long long (*bpf_get_current_pid_tgid)(void) =
36         (void *) BPF_FUNC_get_current_pid_tgid;
37 static unsigned long long (*bpf_get_current_uid_gid)(void) =
38         (void *) BPF_FUNC_get_current_uid_gid;
39 static int (*bpf_get_current_comm)(void *buf, int buf_size) =
40         (void *) BPF_FUNC_get_current_comm;
41 static unsigned long long (*bpf_perf_event_read)(void *map,
42                                                 unsigned long long flags) =
43         (void *) BPF_FUNC_perf_event_read;
44 static int (*bpf_clone_redirect)(void *ctx, int ifindex, int flags) =
45         (void *) BPF_FUNC_clone_redirect;
```

# BPF超能力- 氚石- 限制

---

- eBPF 程序不能调用任意的内核参数，只限于内核模块中列出的 BPF Helper 函数，函数支持列表也随着内核的演进在不断增加。
- eBPF程序不允许包含无法到达的指令，防止加载无效代码，延迟程序的终止。
- eBPF 程序中循环次数限制且必须在有限时间内结束
- 更多信息可以查看：
  - [https://github.com/DavadDi/bpf\\_study#23-ebpf-%E7%9A%84%E9%99%90%E5%88%B6](https://github.com/DavadDi/bpf_study#23-ebpf-%E7%9A%84%E9%99%90%E5%88%B6)

# BPF应用场景

---

- Cilium：完全基于eBPF程序实现了kube-proxy的所有功能，无需依赖iptables和IPVS
- Falco，基于eBPF程序监控和追踪K8S应用的异常行为，并能阻止并发出告警
- Kubectl-trace，一款基于bpftrace的kubectl插件，帮助用户追踪排查K8S应用的运行情况

# BPF社区和生态

---

社区网站，学习BPF的好去处

- <https://ebpf.io> 最全BPF学习资源网站
- [https://lwn.net/Kernel/Index/#Berkeley\\_Packet\\_Filter](https://lwn.net/Kernel/Index/#Berkeley_Packet_Filter) lwn是学习Linux内核技术的最好的网站
- <https://cilium.readthedocs.io/en/stable/bpf/> Cilium提供的BPF文档，是我看到过的最具实战价值的BPF手册
- [https://www.kernel.org/doc/html/latest/bpf/bpf-devel\\_QA.html](https://www.kernel.org/doc/html/latest/bpf/bpf-devel_QA.html) 开发BPF必读Q&A

# BPF社区和生态

---

学习技术还是得从源代码开始

- <https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf.git/> 用于修复bug
- <https://git.kernel.org/pub/scm/linux/kernel/git/bpf/bpf-next.git/> 用于新功能或优化

最近的commits里，不乏有国人的贡献，感兴趣的话，来参与吧～

学习技术也需要沟通交流

- [#ebpf thread of Cilium Slack](https://cilium.slack.com/archives/C4XCTGYEM)
- [https://github.com/DavadDi/bpf\\_study](https://github.com/DavadDi/bpf_study) 狄卫华老师的收集的BPF文章
- <https://github.com/nevermosby/linux-bpf-learning> 本人编写的BPF教程



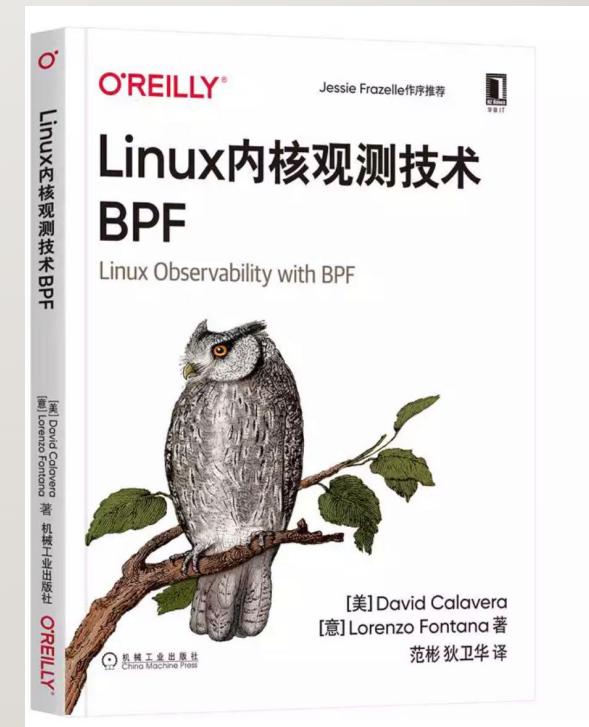
# BPF社区和生态- BPF大神们

- Brendan Gregg, 最强BPF布道师
- Alexei Starovoitov, eBPF创造者
- Daniel Borkmann, eBPF kernel co-maintainer
- Thomas Graf, Cilium之父
- Quentin Monnet, BPFTool co-maintainer

# BPF社区和生态

## BPF书籍

- [Linux Observability with BPF](#)  
David Calavera, Lorenzo Fontana, O'Reilly, Nov 2019
- [Linux内核观测技术BPF](#)  
范彬, 狄卫华译, 2020年8月
- [Systems Performance: Enterprise and the Cloud, 2nd Edition](#)  
Brendan Gregg, Addison-Wesley Professional Computing Series, 2020
- [BPF Performance Tools](#)  
Brendan Gregg, Addison-Wesley Professional Computing Series, Dec 2019



# BPF社区和生态

---

- Kubecon上的BPF相关Session
- LPC 2020 Networking and BPF Summit

# BPF社区和生态

---

## 使用BPF技术的项目

- Tcpdump
- BCC, BPTrace, kubectl-trace from IOVisor
- Cilium from Isovalent
- Falco from Sysdig
- Katran from Facebook
- Bottlerocket from Amazon

- 腾讯云IPVS-BPF K8S网络优化方案
- Kernel Chaos With BPF by PingCAP
- 网易轻舟做系统检测和网络优化
- 字节跳动做高性能网络ACL管理

# 我是如何学习**BPF**技术

---

- 由于对于Cilium感兴趣，半年前开始深入学习BPF技术
- 看文章，编写自己的BPF程序，调试代码，获取期待结果
- 通过写博客总结学习成果

# 我是如何学习**BPF**技术-做计划是全盘思考

---

- 学习之初要做学习计划
- 做计划的过程是我认为就是全盘思考的过程
- 分阶段去完成周期性目标
- 计划不是一成不变的，可以按需进行动态调整

# 我是如何学习BPF技术-不达目的不罢休

---

- 切入口：网络层面的使用场景，XDP、TC、Socket，最终的目标是读懂Cilium BPF代码
- 实践实践再实践
  - 准备环境：vagrant + ubuntu 18.04 with kernel v4.15
  - 写代码
    - XDP程序
    - TC程序
    - BPF Map使用
  - 调试代码
    - 深入了解bpf\_trace\_printk()函数
- 博客
  - 5篇，理论学习加上动手实验

# 我是如何学习BPF技术-阅读理解代码TIPS

---

- 在线阅读Linux内核代码的好去处：<https://elixir.bootlin.com/linux/v5.8.7/source>
  - 快速定位函数的定义和引用
- 下载Linux内核代码，编译运行BPF示例程序
  - 参见博文：<https://davidlovezoe.club/compile-bpf-examples>
- 根据示例程序，写自己的BPF程序，并跑起来

# 我是如何学习**BPF**技术-收获

---

- 静下心来看Linux内核代码
- 理解系统调用、文件系统等功能模块
- 写文章的同时，可以锻炼很多其他软技能，比如画图，录视频，做视频等等，写技术博客就是这么一件痛并快乐着的事情