第二届 eBPF开发者大会

www.ebpftravel.com

eBPF&内核模块在Linux诊断中的应用

银河雷神特大型自研操作系统作者--谢宝友

中国·西安

1、 eBPF&内核模块优劣势

2、 eBPF案例

3、 内核模块案例

1、eBPF优势

✓高安全性

严格验证：确保无死循环、非法内存访问、栈溢出、除数为0等不安全操作，不会崩溃，不会对系统造成损害。
隔离：与内核其他组件隔离，防止未经授权访问内核内存、数据结构和内核源代码。
有限操作：一个受限的指令集。限制可以执行的操作，降低了安全漏洞的风险。

✓高性能

针对特定硬件优化：生成硬件专用指令集，作为本机机器代码运行。
无上下文切换：在内核运行，直接访问内核数据结构和资源。
事件驱动：仅在响应事件时运行，不是一直运行。

✓高扩展性

TCP 拥塞算法：BPF_PROG_TYPE_STRUCT_OPS
extFUSE：用户空间文件系统扩展框架
调度器：BPF_PROG_TYPE_SCHED

✓高可移植性

BPF CO-RE(Compile Once – Run Everywhere)，BTF为内核而设计，帮助eBPF程序兼容不同的内核版本。

1、eBPF劣势

- ✓ **受限于linux及较新内核**
  - 内核版本低于4.14将无法运行
  - 部分新特性在较高版本才支持，并不断演进中
- ✓ **访问操作系统功能是有限的**
  - eBPF的安全性是通过限制程序可访问资源来实现的
  - 其访问操作系统的功能受限
- ✓ **不保证精确计数**
- ✓ **不宜用于频繁执行的函数**

1、内核模块优势

- ✓ 自由调用内核API，甚至私有函数
- ✓ 性能更高
- ✓ 代码尺寸不受限制，可以实现复杂功能
- ✓ 更灵活的挂接钩子
- ✓ 几乎可以用于所有内核版本

1、内核模块劣势

- ✓ 需要小心的编写代码，稳定性验证周期长
- ✓ 版本适配工作更繁琐
- ✓ 稳定性责任划分问题
- ✓ 人才问题

1、 eBPF&内核模块优劣势

2、 eBPF案例

3、 内核模块案例

2、如何避免编译eBPF?

✓ BCC等工具在实践中的缺点

部署时需携带Clang和LLVM等基础库

运行时编译BPF代码，耗费CPU资源，开销高

需发布源码

✓ 解决方案

采用BPF CO-RE技术，在本地主机上预编译BPF程序

- 只需编译机安装Clang和LLVM等基础库，目标机无需部署
- 避免了运行时编译开销高的缺点
- 无需源码发布
- 需要特定版本的内核，并打开相关配置选项

不一样的libbpf

- tools/lib/bpf、tools/bpf

2、如何避免暴露eBPF代码?

- ✓ **核心思想:**

  eBPF所有操作均封装在动态库(.so)中,需要时可以动态获取

  将eBPF程序的load和attach一系列操作流程进行封装，对用户仅暴露必要、最少接口

  注意放宽内核版本匹配

- ✓ **优点:**

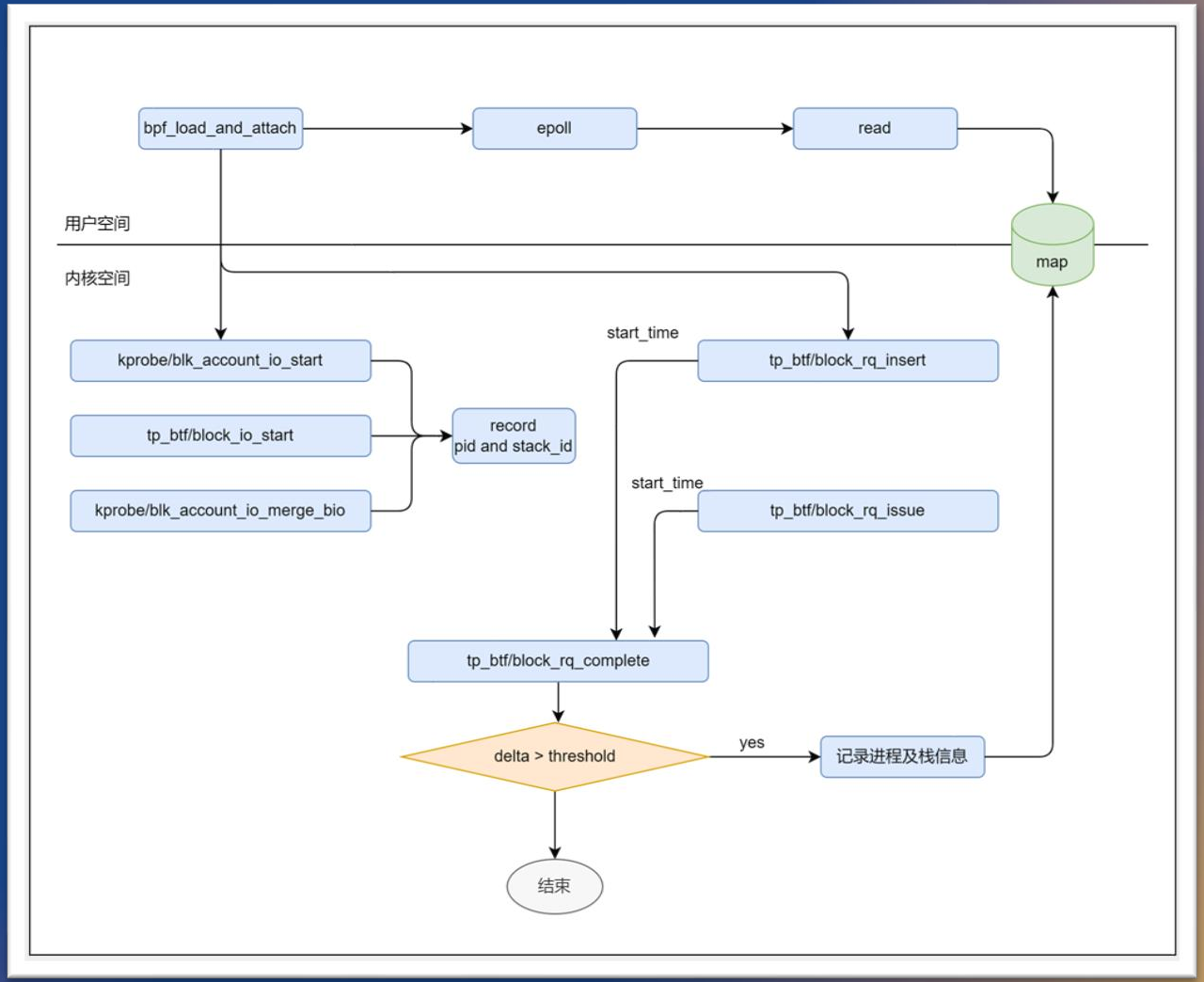
  避免了eBPF代码的暴露

  使用简便，对用户更友好

  避免安装基础库，引起版本冲突并触发生产故障

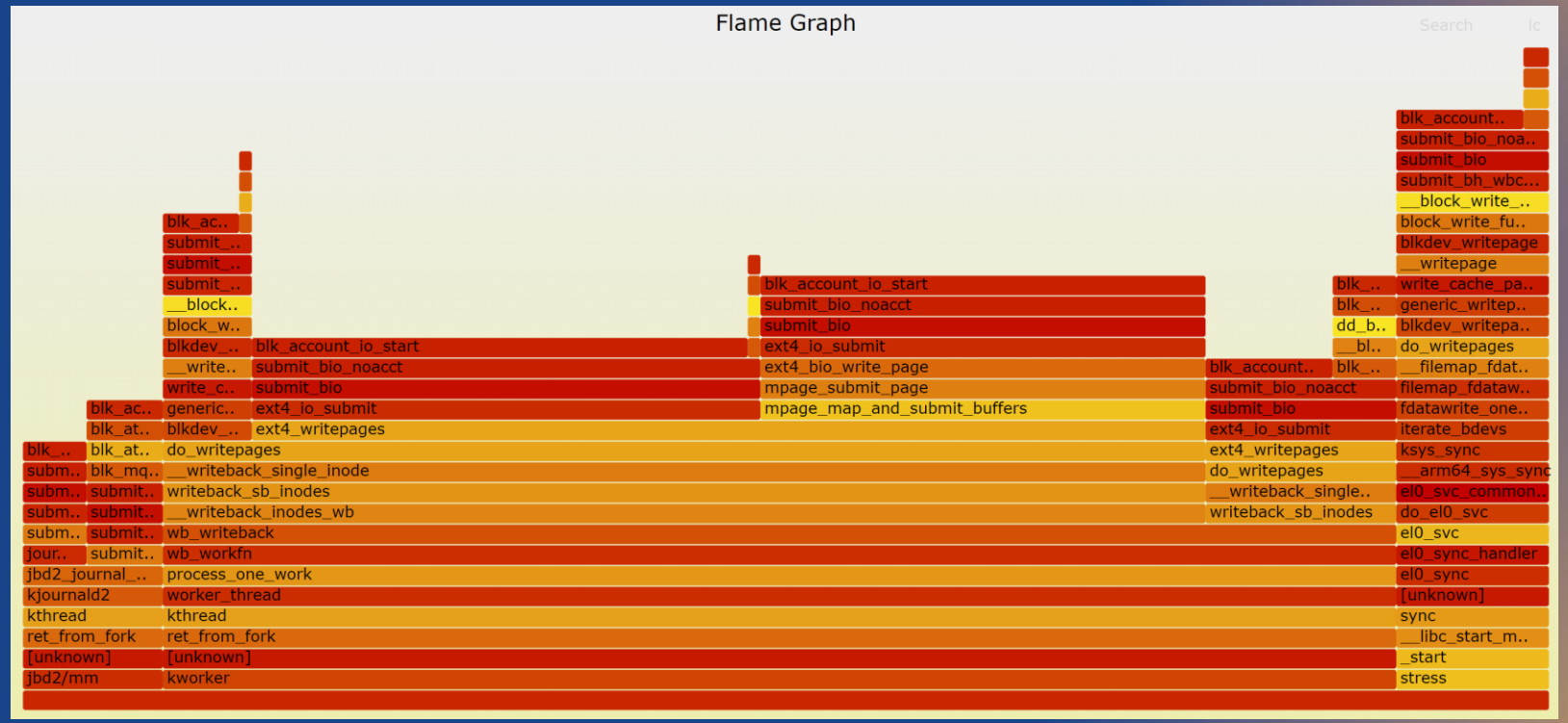
2、eBPF应用案例 --IO延迟

- ✓ 按进程PID，诊断块设备的IO操作的延迟情况
- ✓ 支持用户指定时间阈值
- ✓ 记录堆栈、进程、io、延迟等信息
- ✓ 基于kprobe、tracepoint



2、eBPF应用案例 --IO延迟

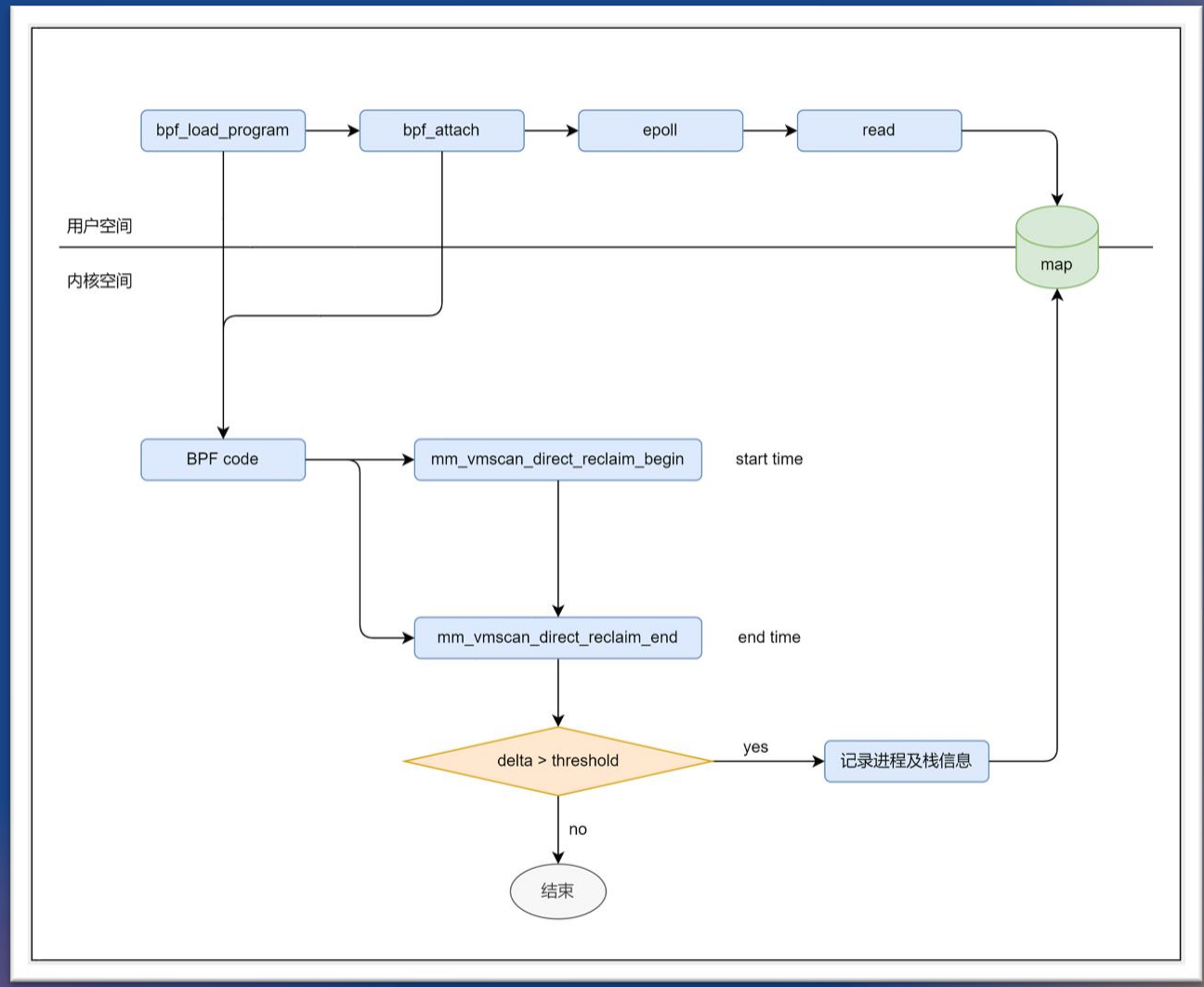
```
[                    ]# chushi-tools bio-snoop -o
TIME(s)     COMM           PID     DISK    T   SECTOR     BYTES   LAT(ms)
0.000000    stress         1585565 mmcblk0 RA  32284536   40960     4.659
0.031813    jbd2/mm        162     mmcblk0 WS  31948448   28672     2.802
0.033935    jbd2/mm        162     mmcblk0 WS  31948504   4096      1.028
0.043295    stress         1585566 mmcblk0 WS  2099200    4096      1.714
0.047634    stress         1585566 mmcblk0 WSM 2099224    4096      5.899
0.048803    stress         1585566 mmcblk0 WSM 39848072   4096      5.626
0.049887    stress         1585566 mmcblk0 WSM 39848192   4096      2.374
0.050958    stress         1585566 mmcblk0 WSM 39853064   4096      2.271
0.051908    stress         1585566 mmcblk0 WSM 39913736   4096      2.137
0.205569    jbd2/mm        162     mmcblk0 WS  31948512   24576     4.436
0.208170    jbd2/mm        162     mmcblk0 WS  31948560   4096      1.522
0.212970    jbd2/mm        162     mmcblk0 WS  31948568   28672     2.311
0.218278    jbd2/mm        162     mmcblk0 WS  31948624   4096      4.267
0.223095    kworker        1585398 mmcblk0 W   46410240   163840    7.893
0.223322    kworker        1585398 mmcblk0 RM  23070832   4096      5.178
0.231512    kworker        1585398 mmcblk0 W   26839040   24576     6.107
0.250412    kworker        1585398 mmcblk0 W   46410560   98304     22.425
0.254101    kworker        1585398 mmcblk0 W   46410752   53248     22.837
0.272039    kworker        1585398 mmcblk0 WS  46410856   466944    23.633
0.280953    kworker        1585398 mmcblk0 WS  46416920   258048    27.829
```



跟踪所有进程的io延迟情况 - stress模拟

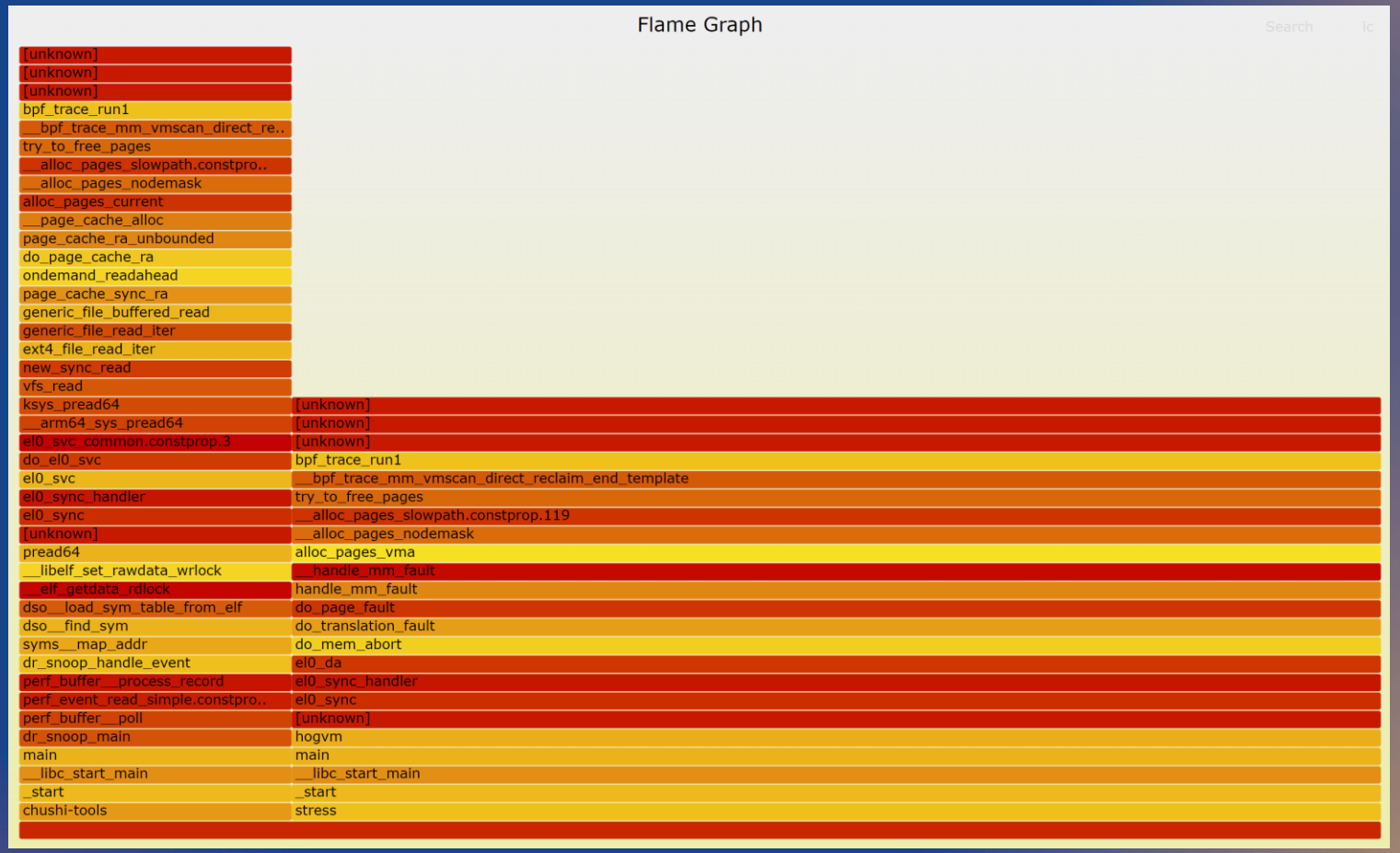
2、eBPF应用案例 --直接内存回收延迟诊断

- ✓ 诊断系统内存直接回收的延时。
- ✓ 支持用户设定时间阈值，
- ✓ 记录进程堆栈、进程ID、延迟时间等。
- ✓ 基于tracepoint



2、eBPF应用案例 --直接内存回收延迟诊断

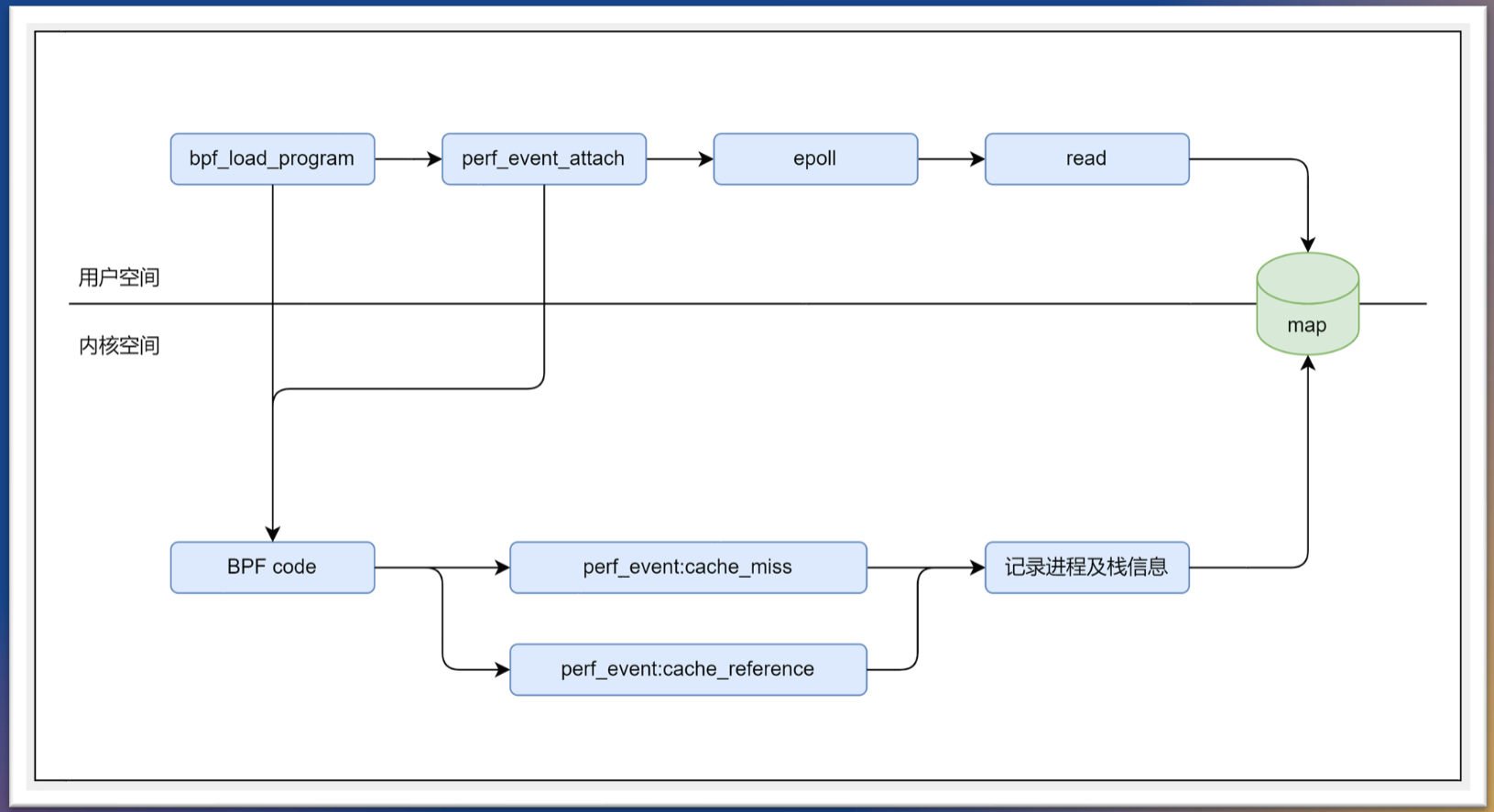
```
[                    ]# chushi-tools dr-snoop -o
Tracing direct reclaim events... Hit Ctrl-C to end.
TIME     COMM             TID     LAT(ms) PAGES
16:00:18 stress           1585687   24.093    41
16:00:18 stress           1585688   35.404    35
16:00:18 stress           1585686   40.405    35
16:00:18 stress           1585687   29.489    60
16:00:18 stress           1585684   66.983    66
16:00:18 stress           1585685   79.700    58
16:00:18 stress           1585688   14.511    37
16:00:18 stress           1585683  100.124    45
16:00:18 stress           1585686   42.319    43
16:00:18 stress           1585685   20.768    33
16:00:18 stress           1585681  103.535    72
16:00:18 stress           1585687   64.113    76
16:00:18 stress           1585682  121.709    77
16:00:18 stress           1585684   77.207    43
16:00:18 stress           1585685   14.501    49
16:00:18 stress           1585688   56.718    82
16:00:18 stress           1585683   51.053    55
```



stress 模拟触发直接内存回收

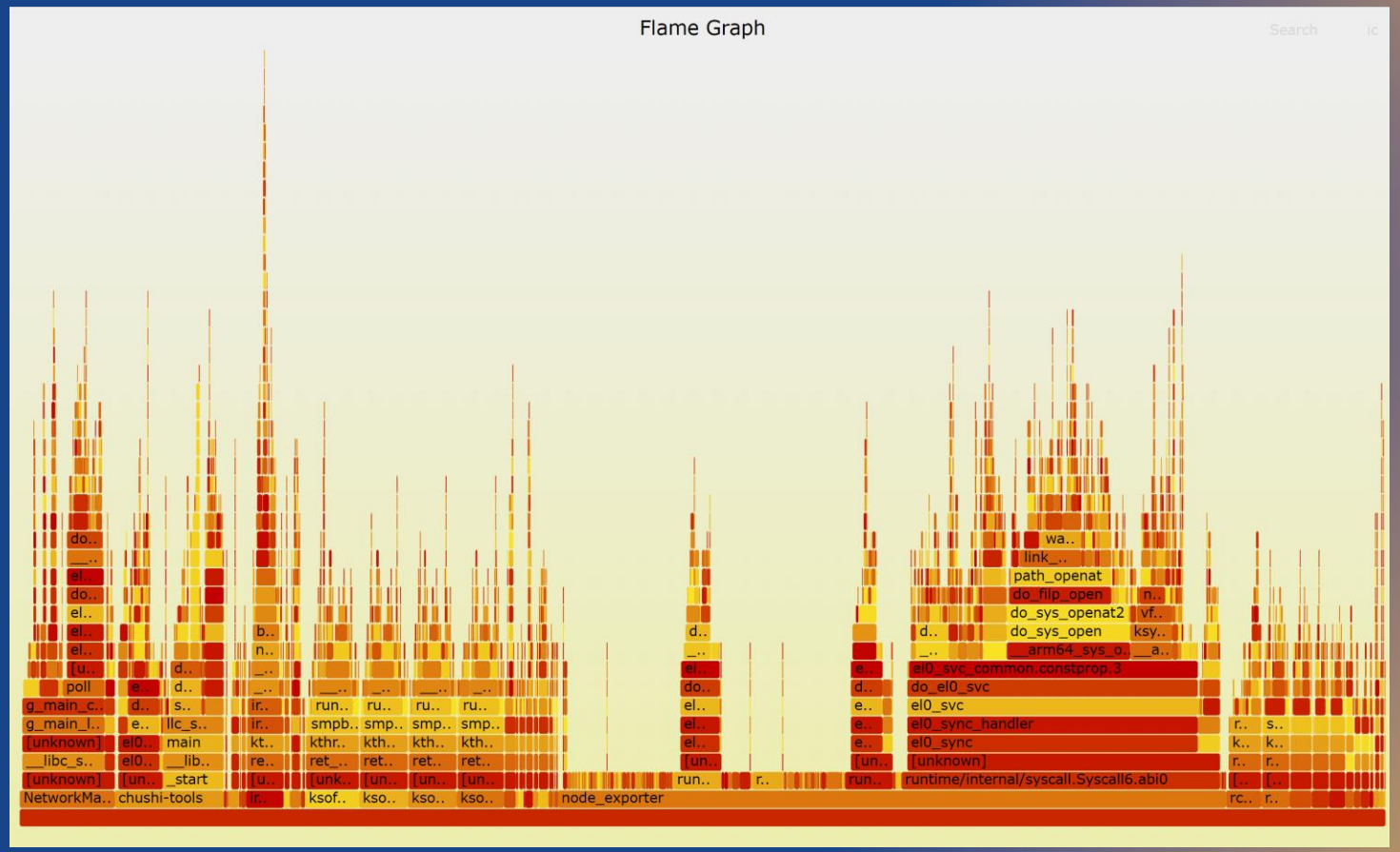
2、eBPF应用案例 -- 缓存命中率

- ✓ 按进程PID统计 CPU 缓存的reference和miss情况。
- ✓ 支持用户设定采样周期、命中率阈值（0~100）
- ✓ 记录命中率低的进程ID和栈信息等；
- ✓ 基于perf



2、eBPF应用案例 -- 缓存命中率

```
[                    ]# chushi-tools llc-stat-bpf -o
Running for 5 seconds or Hit Ctrl-C to end.
PID      NAME             CPU     REFERENCE         MISS     HIT%
448      node_exporter    0               0          100    0.00%
448      node_exporter    1               0          200    0.00%
317      NetworkManager   2               0          100    0.00%
448      node_exporter    0               0          300    0.00%
448      node_exporter    3               0          100    0.00%
317      NetworkManager   2               0          100    0.00%
448      node_exporter    1               0          100    0.00%
448      node_exporter    2               0          100    0.00%
13       rcu_preempt      1               0          100    0.00%
448      node_exporter    1               0          100    0.00%
448      node_exporter    3               0          100    0.00%
448      node_exporter    2               0          100    0.00%
317      NetworkManager   2               0          600    0.00%
448      node_exporter    3               0          100    0.00%
317      NetworkManager   2               0          100    0.00%
448      node_exporter    0               0          100    0.00%
448      node_exporter    0               0          100    0.00%
317      NetworkManager   2               0          300    0.00%
448      node_exporter    0               0          200    0.00%
448      node_exporter    1               0          100    0.00%
448      node_exporter    0               0          100    0.00%
38       ksoftirqd/3      3               0          100    0.00%
448      node_exporter    3               0          100    0.00%
317      NetworkManager   2           13200         4500   65.91%
317      NetworkManager   2               0          500    0.00%
448      node_exporter    1               0          100    0.00%
```



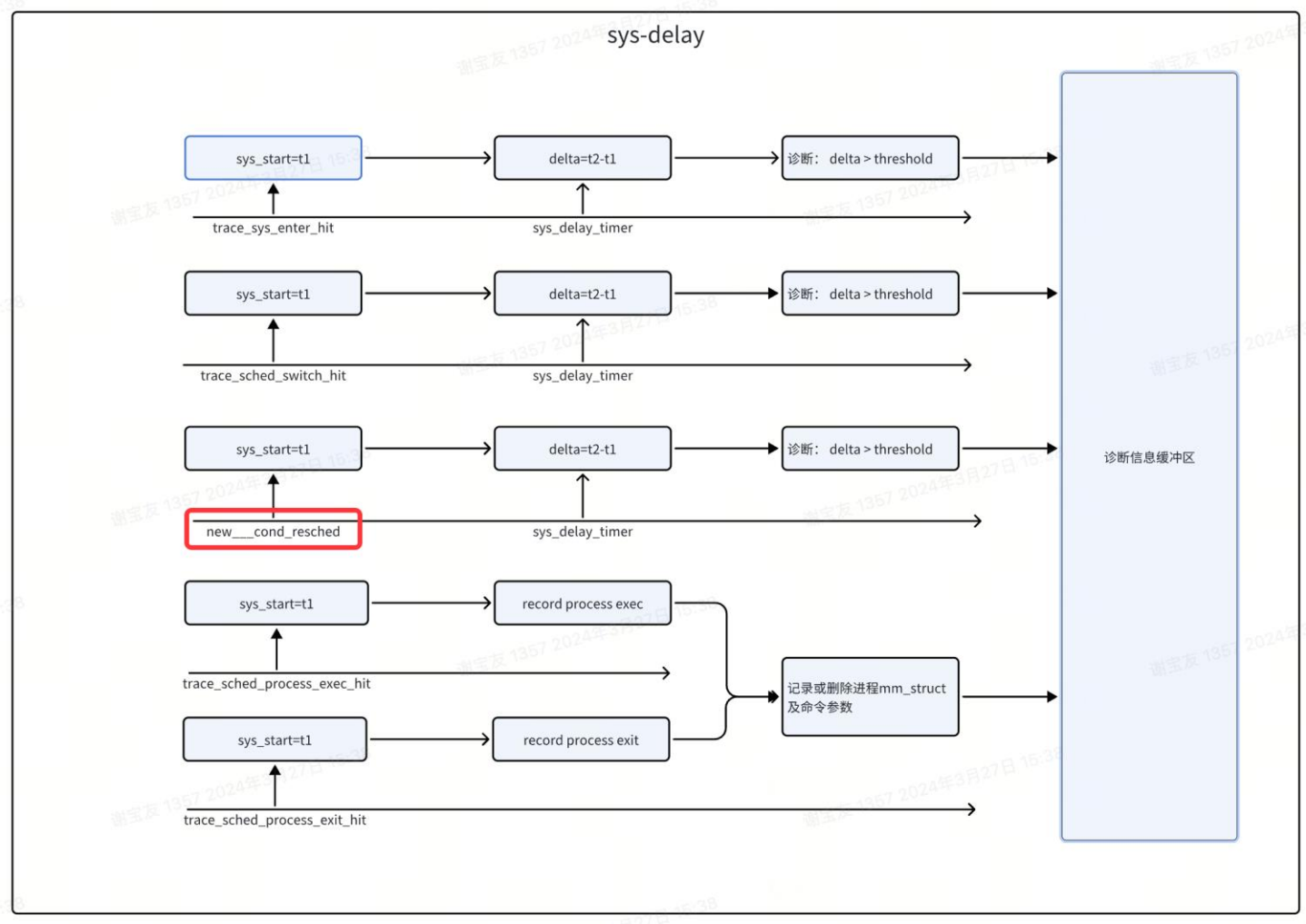
5s内命中率（0~100%）所有进程栈信息

1、 eBPF&内核模块优劣势

2、 eBPF案例

3、 内核模块案例

3、内核模块应用案例




```c
int __sched _cond_resched(void)
{
    if (should_resched()) {
        __cond_resched();
        return 1;
    }
    return 0;
}
EXPORT_SYMBOL(_cond_resched);
```

```c
int new__cond_resched(void *x)
{
    update_sched_time();
    if (should_resched()) {
        atomic64_inc_return(&diag_nr_running);
        orig___cond_resched();
        atomic64_dec_return(&diag_nr_running);
        return 1;
    }
    return 0;
}
```

3、内核模块应用案例

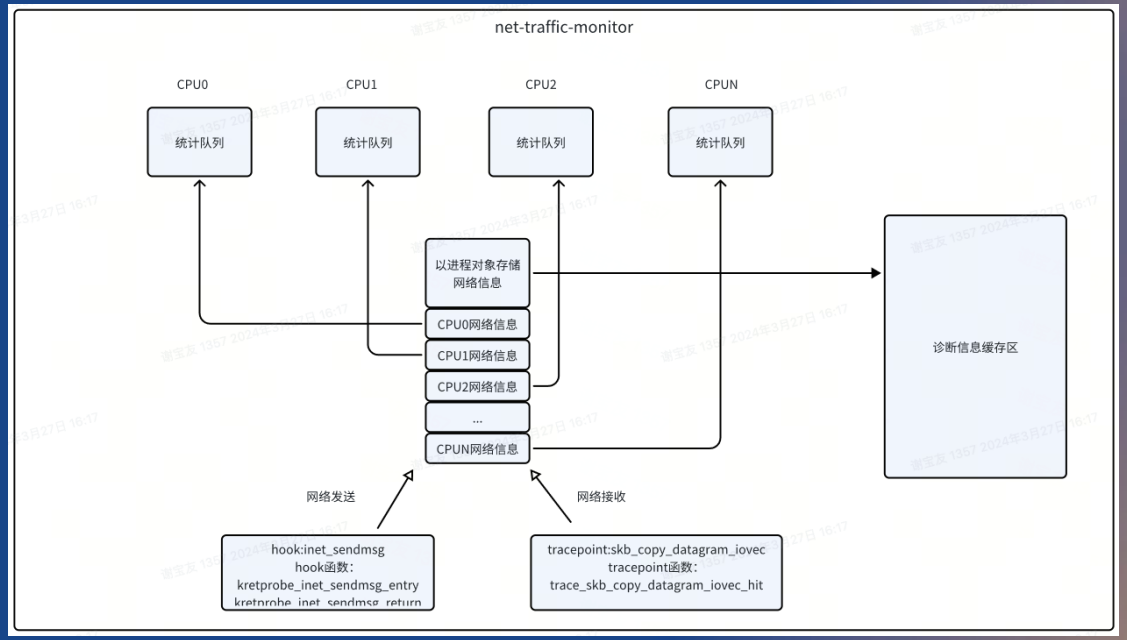
功能：统计应用的域间网络流量

难点：

- IP/域映射关系复杂
- 流量大，每秒10Gb+
- 秒级精确统计

实现要点：

- 每CPU的统计队列
- 小心维护IP/域映射关系，快速查找
- 用户态/内核态数据交换技术



3、内核模块应用案例

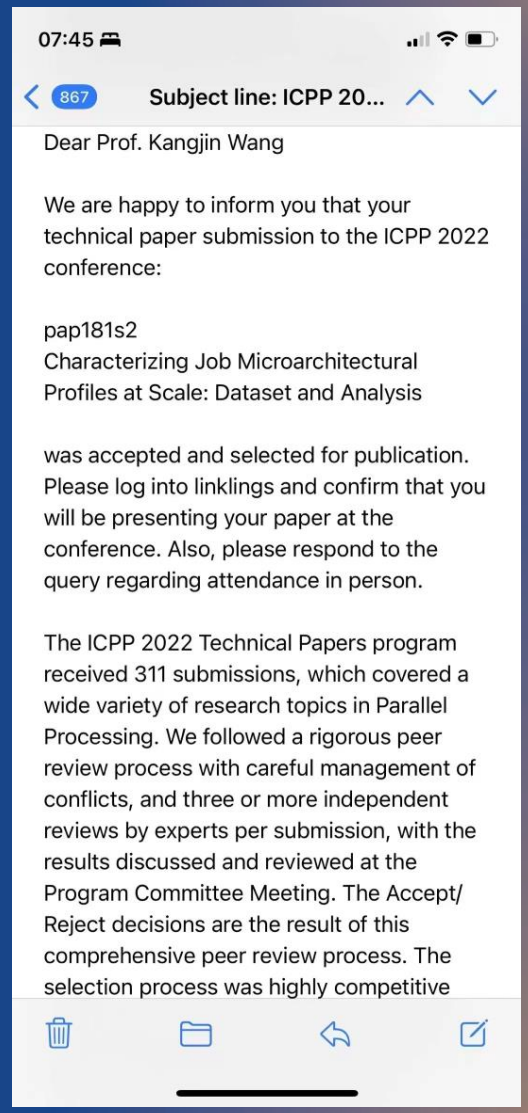
功能：混部应用干扰分析

难点：

- 调度钩子执行频繁
- 按应用/容器统计硬件指标
- 秒级精确统计

实现要点：

- 修复perf子系统缺陷，不重启系统
- IPI / workqueue 汇总数据?
- 短生命周期的容器



不同凡响的银河雷神操作系统