Linux 4.x Tracing: Performance Analysis with bcc/BPF

Brendan Gregg
Senior Performance Architect
Mar 2017
Linux tracing
in the last 3 years...
How do we use these superpowers?
Take aways

1. Understanding the value of Linux tracing superpowers
2. Upgrade to Linux 4.4+ (4.9 is better)
3. Ask for eBPF support in your perf analysis/monitoring tools
Ye Olde BPF
Berkeley Packet Filter

# tcpdump host 127.0.0.1 and port 22 -d

Optimizes packet filter performance

2 x 32-bit registers & scratch memory

User-defined bytecode executed by an in-kernel sandboxed virtual machine

Steven McCanne and Van Jacobson, 1993
Enhanced BPF
aka eBPF or just "BPF"

```c
struct bpf_insn prog[] = {
  BPF_MOV64_REG(BPF_REG_6, BPF_REG_1),
  BPF_LD_ABS(BPF_REG_B, ETH_HLEN + offsetof(struct iphdr, protocol) /* R0 = ip->proto */),
  BPF_STX_MEM(BPF_REG_W, BPF_REG_10, BPF_REG_0, -4), /* *(u32 *)(fp - 4) = r0 */
  BPF_MOV64_REG(BPF_REG_2, BPF_REG_10),
  BPF_ALU64_IMM(BPF_REG_ADD, BPF_REG_2, -4), /* r2 = fp - 4 */
  BPF_LD_MAP_FD(BPF_REG_1, map_fd),
  BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_map_lookup_elem),
  BPF_JMP_IMM(BPF_JEQ, BPF_REG_0, 0, 2),
  BPF_MOV64_IMM(BPF_REG_1, 1), /* r1 = 1 */
  BPF_RAW_INSN(BPF_STX | BPF_XADD | BPF_ADD, BPF_REG_0, BPF_REG_1, 0, 0), /* xadd r0 += r1 */
  BPF_MOV64_IMM(BPF_REG_0, 0), /* r0 = 0 */
  BPF_EXIT_INSN(),
};
```

10 x 64-bit registers
maps (hashes)
actions

Alexei Starovoitov, 2014+
Enhanced BPF Use Cases

User-Defined BPF Programs
- SDN Configuration
- DDoS Mitigation
- Intrusion Detection
- Container Security
- Observability

Kernel
- Runtime
  - verifier
  - BPF actions
- Event Targets
  - sockets
  - kprobes
  - uprobes
  - tracepoints
  - perf_events
Enhanced BPF is in Linux
Demo
New Observability Tools

- Efficient, production safe, useful metrics:

```bash
# biolatency -mT 1
Tracing block device I/O... Hit Ctrl-C to end.

06:20:16

<table>
<thead>
<tr>
<th>msecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>36</td>
<td>***********</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>1</td>
<td>*</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>3</td>
<td>***</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>17</td>
<td>***********</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>33</td>
<td>***********</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>7</td>
<td>*****</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>6</td>
<td>****</td>
</tr>
</tbody>
</table>

[...]
```

These CLI tools may be useful even if you never use them, as examples of what to implement in GUIs
New Visualizations and GUIs

Eg, Netflix self-service UI:

Should be open sourced; you may also build/buy your own
Introducing enhanced BPF

BPF TRACING
A Linux Tracing Timeline

- 1990’s: Static tracers, prototype dynamic tracers
- 2000: LTT + DProbes (dynamic tracing; not integrated)
- 2004: kprobes (2.6.9)
- 2005: DTrace (not Linux), SystemTap (out-of-tree)
- 2008: ftrace (2.6.27)
- 2009: perf_events (2.6.31)
- 2009: tracepoints (2.6.32)
- 2010-2016: ftrace & perf_events enhancements
- 2012: uprobes (3.5)
- **2014-2017: enhanced BPF patches: supporting tracing events**
- 2016-2017: ftrace hist triggers

also: LTTng, ktap, sysdig, ...
## Linux Events & BPF Support

<table>
<thead>
<tr>
<th>BPF output</th>
<th>Dynamic Tracing</th>
<th>Tracepoints</th>
<th>PMCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux 4.4</td>
<td>Linux 4.7</td>
<td>BPF support</td>
<td>Linux 4.9</td>
</tr>
<tr>
<td>Linux 4.6</td>
<td>Linux 4.3</td>
<td>ext4:</td>
<td>cycles</td>
</tr>
<tr>
<td>Linux 4.3</td>
<td></td>
<td>ext4:</td>
<td>instructions</td>
</tr>
<tr>
<td>Linux 4.1</td>
<td></td>
<td>sock:</td>
<td>branch-*</td>
</tr>
<tr>
<td>(version BPF support arrived)</td>
<td></td>
<td>sched:</td>
<td>L1-*</td>
</tr>
<tr>
<td></td>
<td></td>
<td>task:</td>
<td>LLC-*</td>
</tr>
<tr>
<td></td>
<td>kprobes Linux 4.1</td>
<td>signal:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>timer:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>workqueue:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CPU Interconnect</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>kmem:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>vmscan:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>writeback:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>irq:</td>
<td></td>
</tr>
</tbody>
</table>

### Software Events - Linux 4.9
- cpu-clock
- page-faults
- minor-faults
- major-faults
- cs migrations

### Diagram
- Operating System
- Applications
- System Libraries
- System Call Interface
- VFS
- Sockets
- TCP/UDP
- IP
- Ethernet
- Device Drivers
- Block Device Interface
- File Systems
- Volume Manager
- jbd2:
- block:
- scsi:
- skb:
- net:
- sock:
- ext4:
- sock:
- syscalls:
- sched:
- task:
- signal:
- timer:
- workqueue:
- CPU Interconnect
- kmem:
- vmscan:
- writeback:
- irq:
- CPU 1
- Memory Bus
- DRAM
- mem-load
- mem-store
Event Tracing Efficiency

E.g., tracing TCP retransmits

**Old way:** packet capture

- tcpdump
  1. read
  2. dump

- Analyzer
  1. read
  2. process
  3. print

**New way:** dynamic tracing

- Tracer
  1. configure
  2. read

- Kernel
  - buffer
  - file system
  - send
  - receive
  - disks
  - tcp_retransmit_skb()
BPF Tracing Internals

Observability Program

- BPF program
- event config
- per-event data
- statistics
- BPF bytecode

Kernel

- verifier
- BPF
- maps
- load
- attach
- async copy

static tracing
- tracepoints
- kprobes
- uprobes
- sampling, PMCs
- perf_events
Introducing bcc

BPF COMPILER COLLECTION
• BPF Compiler Collection
  – https://github.com/iovisor/bcc
  – Lead developer: Brenden Blanco

• Includes tracing tools

• Provides BPF front-ends:
  – Python
  – Lua
  – C++
  – C helper libraries
  – golang (gobpf)

Tracing layers:
struct bpf_insn prog[] = {
    BPF_MOV64_REG(BPF_REG_6, BPF_REG_1),
    BPF_LD_ABS(BPF_B, ETH_HLEN + offsetof(struct iphdr, protocol) /* R0 = ip->proto*/),
    BPF_STX_MEM(BPF_W, BPF_REG_10, BPF_REG_0, -4), /* *(u32 *)(fp - 4) = r0 */
    BPF_MOV64_REG(BPF_REG_2, BPF_REG_10),
    BPF_ALU64_IMM(BPF_ADD, BPF_REG_2, -4), /* r2 = fp - 4 */
    BPF_LD_MAP_FD(BPF_REG_1, map_fd),
    BPF_RAW_INSNS(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_map_lookup_elem),
    BPF_JMP_IMM(BPF_JEQ, BPF_REG_0, 0, 2),
    BPF_MOV64_IMM(BPF_REG_1, 1), /* r1 = 1 */
    BPF_RAW_INSNS(BPF_STX | BPF_XADD | BPF_DW, BPF_REG_0, BPF_REG_1, 0, 0), /* xadd r0 += r1 */
    BPF_MOV64_IMM(BPF_REG_0, 0), /* r0 = 0 */
    BPF_EXIT_INSNS(),
};
C/BPF

SEC("kprobe/__netif_receive_skb_core")
int bpf_prog1(struct pt_regs *ctx)
{
    /* attaches to kprobe netif_receive_skb,
     * looks for packets on loobpack device and prints them
     */
    char devname[IFNAMSIZ];
    struct net_device *dev;
    struct sk_buff *skb;
    int len;

    /* non-portable! works for the given kernel only */
    skb = (struct sk_buff *) PT_REGS_PARM1(ctx);
    dev = __sk_buff(skb)->dev;
}
# load BPF program
b = BPF(text=""
#include <uapi/linux/ptrace.h>
#include <linux/blkdev.h>
BPF_HISTOGRAM(dist);
int kprobe__blk_account_io_completion(struct pt_regs *ctx,
    struct request *req)
{
    dist.increment(bpf_log2l(req->__data_len / 1024));
    return 0;
}
""")

# header
print("Tracing... Hit Ctrl-C to end.")

# trace until Ctrl-C
try:
    sleep(99999999)
except KeyboardInterrupt:
    print

# output
b["dist"].print_log2_hist("kbytes")

bcc examples/tracing/bitehist.py
entire program
ply/BPF

```plaintext
kretprobe:Sys_read
{
    @.quantize(retval());
}
```

https://github.com/iovisor/ply/blob/master/README.md

entire program
The Tracing Landscape, Mar 2017

(my opinion)

Ease of use

(less brutal)

(sysdig)

(perf)

(ktap)

(dtrace4L.)

(ply/BPF)

(LTTng)

(ftrace)

(bcc/BPF)

(C/BPF)

(Raw BPF)

Scope & Capability

Stage of Development

(alpha) (mature)

recent changes

(hist triggers)

(brutal)

Recent changes

(many)

(less brutal)
For end-users

PERFORMANCE ANALYSIS WITH BCC/BPF
Pre-BPF: Linux Perf Analysis in 60s

1. uptime
2. dmesg -T | tail
3. vmstat 1
4. mpstat -P ALL 1
5. pidstat 1
6. iostat -xz 1
7. free -m
8. sar -n DEV 1
9. sar -n TCP,ETCP 1
10. top

bcc Installation

• https://github.com/iovisor/bcc/blob/master/INSTALL.md
• eg, Ubuntu Xenial:

```bash
# echo "deb [trusted=yes] https://repo.iovisor.org/apt/xenial xenial-nightly main" | \n  sudo tee /etc/apt/sources.list.d/iovisor.list
# sudo apt-get update
# sudo apt-get install bcc-tools
```

  – Also available as an Ubuntu snap
  – Ubuntu 16.04 is good, 16.10 better: more tools work

• Installs many tools
  – In /usr/share/bcc/tools, and .../tools/old for older kernels
bcc General Performance Checklist

1. execsnoop
2. opensnoop
3. ext4slower(...)
4. biolatency
5. biosnoop
6. cachestat
7. tcpconnect
8. tcpaccept
9. tcpretrans
10. gethostlatency
11. runqlat
12. profile

https://github.com/iovisor/bcc#tools.2017
1. execsnoop

- Trace new processes and debug short-lived process issues:

<table>
<thead>
<tr>
<th># execsnoop</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCOMM</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>bash</td>
</tr>
<tr>
<td>preconv</td>
</tr>
<tr>
<td>man</td>
</tr>
<tr>
<td>man</td>
</tr>
<tr>
<td>man</td>
</tr>
<tr>
<td>nroff</td>
</tr>
<tr>
<td>nroff</td>
</tr>
<tr>
<td>groff</td>
</tr>
<tr>
<td>groff</td>
</tr>
</tbody>
</table>

[...]

Efficient: only traces exec()
2. opensnoop

- Find config, log, and data files, and inefficient file usage:

<table>
<thead>
<tr>
<th>#</th>
<th>opensnoop</th>
<th>FD</th>
<th>ERR</th>
<th>PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>27159</td>
<td>catalina.sh</td>
<td>3</td>
<td>0</td>
<td>/apps/tomcat8/bin/setclasspath.sh</td>
</tr>
<tr>
<td>4057</td>
<td>redis-server</td>
<td>5</td>
<td>0</td>
<td>/proc/4057/stat</td>
</tr>
<tr>
<td>2360</td>
<td>redis-server</td>
<td>5</td>
<td>0</td>
<td>/proc/2360/stat</td>
</tr>
<tr>
<td>30668</td>
<td>ssdh</td>
<td>4</td>
<td>0</td>
<td>/proc/sys/kernel/ngroups_max</td>
</tr>
<tr>
<td>30668</td>
<td>ssdh</td>
<td>4</td>
<td>0</td>
<td>/etc/group</td>
</tr>
<tr>
<td>30668</td>
<td>ssdh</td>
<td>4</td>
<td>0</td>
<td>/root/.ssh/authorized_keys</td>
</tr>
<tr>
<td>30668</td>
<td>ssdh</td>
<td>-1</td>
<td>2</td>
<td>/var/run/nologin</td>
</tr>
<tr>
<td>30668</td>
<td>ssdh</td>
<td>-1</td>
<td>2</td>
<td>/etc/nologin</td>
</tr>
<tr>
<td>30668</td>
<td>ssdh</td>
<td>4</td>
<td>0</td>
<td>/etc/login.defs</td>
</tr>
<tr>
<td>30668</td>
<td>ssdh</td>
<td>4</td>
<td>0</td>
<td>/etc/passwd</td>
</tr>
</tbody>
</table>

[...] Like "strace -feopen", but system-wide and low overhead
3. ext4slower

- Trace slow FS I/O, to better identify I/O issues and outliers:

```bash
# ext4slower 1
Tracing ext4 operations slower than 1 ms

<table>
<thead>
<tr>
<th>TIME</th>
<th>COMM</th>
<th>PID</th>
<th>T</th>
<th>BYTES</th>
<th>OFF_KB</th>
<th>LAT(ms)</th>
<th>FILENAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>06:49:17</td>
<td>bash</td>
<td>3616</td>
<td>R</td>
<td>128</td>
<td>0</td>
<td>7.75</td>
<td>cksum</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>39552</td>
<td>0</td>
<td>1.34</td>
<td>[</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>96</td>
<td>0</td>
<td>5.36</td>
<td>2to3-2.7</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>96</td>
<td>0</td>
<td>14.94</td>
<td>2to3-3.4</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>10320</td>
<td>0</td>
<td>6.82</td>
<td>411toppm</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>65536</td>
<td>0</td>
<td>4.01</td>
<td>a2p</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>55400</td>
<td>0</td>
<td>8.77</td>
<td>ab</td>
</tr>
<tr>
<td>06:49:17</td>
<td>cksum</td>
<td>3616</td>
<td>R</td>
<td>36792</td>
<td>0</td>
<td>16.34</td>
<td>aclocal-1.14</td>
</tr>
</tbody>
</table>
```

More reliable and complete indicator than measuring disk I/O latency
Also: btrfslower, xfslower, zfsslower
4. biolatency

- Identify multimodal latency and outliers with a histogram:

```sh
# biolatency -mT 1
Tracing block device I/O... Hit Ctrl-C to end.

06:20:16

<table>
<thead>
<tr>
<th>msecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>36</td>
<td>**************************************</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>1</td>
<td>*</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>3</td>
<td>***</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>17</td>
<td>*****************</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>33</td>
<td>******************************************</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>7</td>
<td>********</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>6</td>
<td>*****</td>
</tr>
</tbody>
</table>

[...]
```

Average latency (iostat/sar) may not be representative with multiple modes or outliers.

The "count" column is summarized in-kernel.
5. biosnoop

- Dump disk I/O events for detailed analysis. tcpdump for disks:

<table>
<thead>
<tr>
<th># biosnoop</th>
<th>TIME(s)</th>
<th>COMM</th>
<th>PID</th>
<th>DISK</th>
<th>T</th>
<th>SECTOR</th>
<th>BYTES</th>
<th>LAT(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.000004001</td>
<td>supervise</td>
<td>1950</td>
<td>xvda1</td>
<td>W</td>
<td>13092560</td>
<td>4096</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>0.000178002</td>
<td>supervise</td>
<td>1950</td>
<td>xvda1</td>
<td>W</td>
<td>13092432</td>
<td>4096</td>
<td>0.61</td>
</tr>
<tr>
<td></td>
<td>0.001469001</td>
<td>supervise</td>
<td>1956</td>
<td>xvda1</td>
<td>W</td>
<td>13092440</td>
<td>4096</td>
<td>1.24</td>
</tr>
<tr>
<td></td>
<td>0.001588002</td>
<td>supervise</td>
<td>1956</td>
<td>xvda1</td>
<td>W</td>
<td>13115128</td>
<td>4096</td>
<td>1.09</td>
</tr>
<tr>
<td></td>
<td>1.022346001</td>
<td>supervise</td>
<td>1950</td>
<td>xvda1</td>
<td>W</td>
<td>13115272</td>
<td>4096</td>
<td>0.98</td>
</tr>
<tr>
<td></td>
<td>1.022568002</td>
<td>supervise</td>
<td>1950</td>
<td>xvda1</td>
<td>W</td>
<td>13188496</td>
<td>4096</td>
<td>0.93</td>
</tr>
<tr>
<td></td>
<td>1.023534000</td>
<td>supervise</td>
<td>1956</td>
<td>xvda1</td>
<td>W</td>
<td>13188520</td>
<td>4096</td>
<td>0.79</td>
</tr>
<tr>
<td></td>
<td>1.023585003</td>
<td>supervise</td>
<td>1956</td>
<td>xvda1</td>
<td>W</td>
<td>13189512</td>
<td>4096</td>
<td>0.60</td>
</tr>
<tr>
<td></td>
<td>2.003920000</td>
<td>xfsaid/md0</td>
<td>456</td>
<td>xvdc</td>
<td>W</td>
<td>62901512</td>
<td>8192</td>
<td>0.23</td>
</tr>
</tbody>
</table>

[...] Can import this into a spreadsheet and do a scatter plot of time vs latency, e.t.c.
6. cachestat

• Measure file system cache hit ratio statistics:

<table>
<thead>
<tr>
<th></th>
<th>cachestat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HITs</td>
</tr>
<tr>
<td>170610</td>
<td>41607</td>
</tr>
<tr>
<td>157693</td>
<td>6149</td>
</tr>
<tr>
<td>174483</td>
<td>20166</td>
</tr>
<tr>
<td>434778</td>
<td></td>
</tr>
<tr>
<td>435723</td>
<td></td>
</tr>
<tr>
<td>846183</td>
<td>83800</td>
</tr>
<tr>
<td>96387</td>
<td></td>
</tr>
<tr>
<td>120258</td>
<td></td>
</tr>
<tr>
<td>255861</td>
<td></td>
</tr>
<tr>
<td>191388</td>
<td></td>
</tr>
</tbody>
</table>

These stats should be added to /proc
7. tcpconnect

- Trace active ("outbound") TCP connections:

<table>
<thead>
<tr>
<th>PID</th>
<th>COMM</th>
<th>IP</th>
<th>SADDR</th>
<th>DADDAR</th>
<th>DPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>25333</td>
<td>recordProgra</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>28527</td>
<td></td>
</tr>
<tr>
<td>25338</td>
<td>curl</td>
<td>100.66.3.172</td>
<td>52.22.109.254</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>25340</td>
<td>curl</td>
<td>100.66.3.172</td>
<td>31.13.73.36</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>25342</td>
<td>curl</td>
<td>100.66.3.172</td>
<td>104.20.25.153</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>25344</td>
<td>curl</td>
<td>100.66.3.172</td>
<td>50.56.53.173</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>25365</td>
<td>recordProgra</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>28527</td>
<td></td>
</tr>
<tr>
<td>26119</td>
<td>ssh</td>
<td>::1</td>
<td>::1</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>25388</td>
<td>recordProgra</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>28527</td>
<td></td>
</tr>
<tr>
<td>25220</td>
<td>ssh</td>
<td>fe80::8a3:9dff::fed5:6b19 fe80::8a3:9dff::fed5:6b19</td>
<td>22</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Efficient: dynamic tracing of TCP connect functions only; does not trace send/receive
8. tcpaccept

- Trace passive ("inbound") TCP connections:

<table>
<thead>
<tr>
<th>PID</th>
<th>COMM</th>
<th>IP</th>
<th>RADDR</th>
<th>LADDR</th>
<th>LPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2287</td>
<td>sshd</td>
<td>4 11.16.213.254</td>
<td>100.66.3.172</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>4057</td>
<td>redis-server</td>
<td>4 127.0.0.1</td>
<td>127.0.0.1</td>
<td>28527</td>
<td></td>
</tr>
<tr>
<td>4057</td>
<td>redis-server</td>
<td>4 127.0.0.1</td>
<td>127.0.0.1</td>
<td>28527</td>
<td></td>
</tr>
<tr>
<td>4057</td>
<td>redis-server</td>
<td>4 127.0.0.1</td>
<td>127.0.0.1</td>
<td>28527</td>
<td></td>
</tr>
<tr>
<td>4057</td>
<td>redis-server</td>
<td>4 127.0.0.1</td>
<td>127.0.0.1</td>
<td>28527</td>
<td></td>
</tr>
<tr>
<td>2287</td>
<td>sshd</td>
<td>6 ::1</td>
<td>::1</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>4057</td>
<td>redis-server</td>
<td>4 127.0.0.1</td>
<td>127.0.0.1</td>
<td>28527</td>
<td></td>
</tr>
<tr>
<td>4057</td>
<td>redis-server</td>
<td>4 127.0.0.1</td>
<td>127.0.0.1</td>
<td>28527</td>
<td></td>
</tr>
<tr>
<td>2287</td>
<td>sshd</td>
<td>6 fe80::8a3:9dff:fed5:6b19</td>
<td>fe80::8a3:9dff:fed5:6b19</td>
<td>22</td>
<td></td>
</tr>
</tbody>
</table>

Efficiently: no send/receive tracing
9. tcpretrans

• Trace TCP retransmits with kernel state (not on the wire):

<table>
<thead>
<tr>
<th>TIME</th>
<th>PID</th>
<th>IP</th>
<th>LADDR:LPORT</th>
<th>T&gt; RADDR:RPORT</th>
<th>STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>01:55:05</td>
<td>0</td>
<td>4</td>
<td>10.153.223.157:22</td>
<td>R&gt; 69.53.245.40:34619</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>01:55:05</td>
<td>0</td>
<td>4</td>
<td>10.153.223.157:22</td>
<td>R&gt; 69.53.245.40:34619</td>
<td>ESTABLISHED</td>
</tr>
</tbody>
</table>

Efficiently: no send/receive tracing
10. gethostlatency

- Trace DNS latency system-wide (it's always DNS):

```
# gethostlatency

<table>
<thead>
<tr>
<th>TIME</th>
<th>PID</th>
<th>COMM</th>
<th>LATms</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>06:10:24</td>
<td>28011</td>
<td>wget</td>
<td>90.00</td>
<td><a href="http://www.iovisor.org">www.iovisor.org</a></td>
</tr>
<tr>
<td>06:10:28</td>
<td>28127</td>
<td>wget</td>
<td>0.00</td>
<td><a href="http://www.iovisor.org">www.iovisor.org</a></td>
</tr>
<tr>
<td>06:10:41</td>
<td>28404</td>
<td>wget</td>
<td>9.00</td>
<td><a href="http://www.netflix.com">www.netflix.com</a></td>
</tr>
<tr>
<td>06:10:48</td>
<td>28544</td>
<td>curl</td>
<td>35.00</td>
<td><a href="http://www.netflix.com.au">www.netflix.com.au</a></td>
</tr>
<tr>
<td>06:11:10</td>
<td>29054</td>
<td>curl</td>
<td>31.00</td>
<td><a href="http://www.plumgrid.com">www.plumgrid.com</a></td>
</tr>
<tr>
<td>06:11:16</td>
<td>29195</td>
<td>curl</td>
<td>3.00</td>
<td><a href="http://www.facebook.com">www.facebook.com</a></td>
</tr>
<tr>
<td>06:11:24</td>
<td>25313</td>
<td>wget</td>
<td>3.00</td>
<td><a href="http://www.usenix.org">www.usenix.org</a></td>
</tr>
<tr>
<td>06:11:25</td>
<td>29404</td>
<td>curl</td>
<td>72.00</td>
<td>foo</td>
</tr>
<tr>
<td>06:11:28</td>
<td>29475</td>
<td>curl</td>
<td>1.00</td>
<td>foo</td>
</tr>
</tbody>
</table>
```

Instruments getaddrinfo(), gethostbyname(), e.t.c.
11. runqlat

- Examine CPU scheduler run queue latency as a histogram:

```
# runqlat -m 5
Tracing run queue latency... Hit Ctrl-C to end.

<table>
<thead>
<tr>
<th>msecs</th>
<th>: count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>: 3818</td>
<td>******************************************</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>: 39</td>
<td></td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>: 39</td>
<td></td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>: 62</td>
<td></td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>: 2214</td>
<td>*****************************************</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>: 226</td>
<td>**</td>
</tr>
</tbody>
</table>

[...]
```

As efficient as possible: scheduler calls can become frequent
12. profile

- An efficient profiler. Count unique stacks in kernel:

```plaintext
# profile
Sampling at 49 Hertz of all threads by user + kernel stack... Ctrl-C to end.
^C
[...]
ffffffff813d0af8 __clear_user
ffffffff813d5277 iov_iter_zero
ffffffff814ec5f2 read_iter_zero
ffffffff8120be9d __vfs_read
ffffffff8120c385 vfs_read
ffffffff8120d786 sys_read
ffffffff817cc076 entry_SYSCALL_64_fastpath
00007fc5652ad9b0 read
- dd (25036)
  7
[...]
```

This stack was sampled 7 times
Advanced Analysis

• Find/draw a functional diagram
• Apply performance methods
  http://www.brendangregg.com/methodology.html
  1. Workload Characterization
  2. Latency Analysis
  3. USE Method
  4. Performance Mantras
• Start with the Q's, then find the A's
• Use multi-tools:
  – funcount, trace, argdist, stackcount

E.g., storage I/O subsystem:
Performance Mantras

1. Don't do it
2. Do it, but don't do it again
3. Do it less
4. Do it later
5. Do it when they're not looking
6. Do it concurrently
7. Do it cheaper

→ eliminate unnecessary work

from Craig Hanson and Pat Crain, and the performance engineering community
trace

# trace 'sys_read (arg3 > 20000) "read %d bytes", arg3'
TIME    PID   COMM       FUNC
05:18:23 4490  dd       sys_read   read 1048576 bytes
05:18:23 4490  dd       sys_read   read 1048576 bytes
05:18:23 4490  dd       sys_read   read 1048576 bytes
^C

# trace -h
[...]
trace -K blk_account_io_start
   Trace this kernel function, and print info with a kernel stack trace
trace 'do_sys_open "%s", arg2'
   Trace the open syscall and print the filename being opened
trace 'sys_read (arg3 > 20000) "read %d bytes", arg3'
   Trace the read syscall and print a message for reads >20000 bytes
trace r::do_sys_return
   Trace the return from the open syscall
trace 'c:open (arg2 == 42) "%s %d", arg1, arg2'
   Trace the open() call from libc only if the flags (arg2) argument is 42
trace 't:block:block_rq_complete "sectors=%d", args->nr_sector'
   Trace the block_rq_complete kernel tracepoint and print # of tx sectors
[...]
```
# argdist -H 'p::tcp_cleanup_rbuf(struct sock *sk, int copied):int:copied'
[15:34:45]
<table>
<thead>
<tr>
<th>copied</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>15088</td>
<td>*********************************************************</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>4786</td>
<td><strong>--------------------</strong></td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>256 -&gt; 511</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>512 -&gt; 1023</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1024 -&gt; 2047</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>2048 -&gt; 4095</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4096 -&gt; 8191</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>8192 -&gt; 16383</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>16384 -&gt; 32767</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
```

by Sasha Goldshtein
Coming to a GUI near you

BCC/BPF VISUALIZATIONS
Latency Heatmaps
CPU + Off-CPU Flame Graphs

- Can now be BPF optimized

[Kernel CPU Flame Graph: Linux build]

[Kernel Off-CPU Time Flame Graph: Linux build]

http://www.brendangregg.com/flamegraphs.html
Conquer Performance

On-CPU + off-CPU means we can measure everything

Except sometimes one off-CPU stack isn't enough...
Off-Wake Flame Graphs

• Shows blocking stack with waker stack
  – Better understand why blocked
  – Merged in-kernel using BPF
  – Include multiple waker stacks == chain graphs

• We couldn't do this before
Overview for tool developers

HOW TO PROGRAM BCC/BPF
bcc Tutorials

2. .../docs/tutorial.md
3. .../docs/tutorial_bcc_python_developer.md
4. .../docs/reference_guide.md
5. .../CONTRIBUTING-SCRIPTS.md
# ./bitehist.py
Tracing... Hit Ctrl-C to end.

^C

<table>
<thead>
<tr>
<th>kbytes</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>211</td>
<td>**********</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>800</td>
<td>****************************************</td>
</tr>
</tbody>
</table>
# load BPF program
b = BPF(text=""
#include <uapi/linux/ptrace.h>
#include <linux/blkdev.h>
BPF_HISTOGRAM(dist);
int kprobe__blk_account_io_completion(struct pt_regs *ctx,
    struct request *req)
{
    dist.increment(bpf_vlog2l(req->__data_len / 1024));
    return 0;
}
""")

# header
print("Tracing... Hit Ctrl-C to end.")

# trace until Ctrl-C
try:
    sleep(99999999)
except KeyboardInterrupt:
    print

# output
b["dist"].print_vlog2_hist("kbytes")
bytehist.py Annotated

```python
# header
print("Tracing... Hit Ctrl-C to end.")

# trace until Ctrl-C
try:
    sleep(99999999)
except KeyboardInterrupt:
    print

# output
b["dist"].print_log2_hist("kbytes")
```

C BPF Program

```c
#include "uapi/linux/ptrace.h"
#include "linux/blkdev.h"

BPF_HISTOGRAM(dist);

int kprobe__blk_account_io_completion(struct pt_regs *ctx, 
    struct request *req) 
{
    dist.increment(bpf_log2l(req->__data_len / 1024));
    return 0;
}
"kprobe__" is a shortcut for BPF.attach_kprobe()
""
```

Event

Map

Python Program

Statistics

`bytehist.py` is a BPF and Python program to trace I/O completion events and print a histogram of the number of bytes transferred.
Current Complications

- Initialize all variables
- Extra bpf_probe_read()s
- BPF_PERF_OUTPUT()
- Verifier errors

```c
struct sock *skp = NULL;
bpf_probe_read(&skp, sizeof(skb), &sk);

// pull in details
u16 family = 0, lport = 0, dport = 0;
char state = 0;
bpf_probe_read(&family, sizeof(family), &skp->__sk_common.s
bpf_probe_read(&lport, sizeof(lport), &skp->__sk_common.s
bpf_probe_read(&dport, sizeof(dport), &skp->__sk_common.s
bpf_probe_read(&state, sizeof(state), (void *)skp->__sk

if (family == AF_INET) {
    struct ipv4_data_t data4 = {.pid = pid, .ip = 4, .typ
    bpf_probe_read(&data4.saddr, sizeof(u32),
        &skp->__sk_common.skc_rcv_saddr);
bpf_probe_read(&data4.daddr, sizeof(u32),
        &skp->__sk_common.skc_daddr);
    // lport is host order
    data4.lport = lport;
data4.dport = ntohs(dport);
data4.state = state;
    ipv4_events.perf_submit(ctx, &data4, sizeof(data4));
```
Bonus Round

PLY
# ply -c 'kprobe:do_sys_open { printf("opened: %s\n", mem(arg(1), "128s")); }'

1 probe active

opened: /sys/kernel/debug/tracing/events/enable
opened: /etc/ld.so.cache
opened: /lib/x86_64-linux-gnu/libselinux.so.1
opened: /lib/x86_64-linux-gnu/libc.so.6
opened: /lib/x86_64-linux-gnu/libpcre.so.3
opened: /lib/x86_64-linux-gnu/libdl.so.2
opened: /lib/x86_64-linux-gnu/libpthread.so.0
opened: /proc/filesystems
opened: /usr/lib/locale/locale-archive
opened: .

[...]
Count vfs calls

```bash
# ply -c 'kprobe:vfs_* { @[func()].count(); }'
WRN kprobe_attach_pattern: 'vfs_caches_init_early' will not be probed
WRN kprobe_attach_pattern: 'vfs_caches_init' will not be probed

49 probes active
^Cde-activating probes

@:
    vfs_fstat       33
    vfs_getattr     34
    vfs_getattr_nosec 35
    vfs_open       49
    vfs_read       66
    vfs_write      68
    [...]
```
Read return size

```
# ply -c 'kretprobe:SyS_read { @ret.quantize(retval()); }'
1 probe active
^Cde-activating probes

@ret:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>[2, 3]</th>
<th>[4, 7]</th>
<th>[8, 15]</th>
<th>[16, 31]</th>
<th>[32, 63]</th>
<th>[64, 127]</th>
<th>[128, 255]</th>
<th>[256, 511]</th>
<th>[512, 1K)</th>
</tr>
</thead>
<tbody>
<tr>
<td>index</td>
<td>0</td>
<td>7</td>
<td>24</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>11</td>
<td>22</td>
<td>10</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>
```


Read return size (ASCII)

```bash
# ply -A -c 'kretprobe:SyS_read { @ret.quantize(retval()); }'
1 probe active
^C de-activating probes

@ret:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>12</td>
</tr>
<tr>
<td>[ 2, 3]</td>
<td>7</td>
</tr>
<tr>
<td>[ 4, 7]</td>
<td>0</td>
</tr>
<tr>
<td>[ 8, 15]</td>
<td>1</td>
</tr>
<tr>
<td>[ 16, 31]</td>
<td>2</td>
</tr>
<tr>
<td>[ 32, 63]</td>
<td>2</td>
</tr>
<tr>
<td>[ 64, 127]</td>
<td>3</td>
</tr>
<tr>
<td>[ 128, 255]</td>
<td>2</td>
</tr>
<tr>
<td>[ 256, 511]</td>
<td>1</td>
</tr>
<tr>
<td>[ 512, 1k)</td>
<td>11</td>
</tr>
</tbody>
</table>
```
Read latency

```bash
# ply -A -c 'kprobe:Sys_read { @start[tid()] = nsecs(); } kretprobe:Sys_read /@start[tid()]/ { @ns.quantize(nsecs() - @start[tid()]); @start[tid()] = nil; }'
```

2 probes active
^Cde-activating probes

```
[512, 1k)   3 |#################
[ 1k, 2k)   7 |#############################################################
[ 2k, 4k)  12 |#################################################################
[ 4k, 8k)   3 |#############################
[ 8k, 16k)  2 |#####
[ 16k, 32k) 0 |
[ 32k, 64k) 0 |
[ 64k, 128k) 3 |#############################
[128k, 256k) 1 |###
[256k, 512k) 1 |##
[512k, 1M)  2 |#####
```

[...]
Counting Off-CPU stacks

```
# ply -c 'kprobe:schedule { @[stack()].count() }'
1 probe active
^Cde-activating probes

@:
    schedule+0x1
    sys_exit+0x17
    do_syscall_64+0x5e
    return_from_SYSCALL_64 1

[...]
    schedule+0x1
    fuse_dev_read+0x63
    new_sync_read+0xd2
    __vfs_read+0x26
    vfs_read+0x96
    sys_read+0x55
    do_syscall_64+0x5e
    return_from_SYSCALL_64 1707

    schedule+0x1
    do_syscall_64+0xa2
    return_from_SYSCALL_64 4647
```
ply One-Liners

# Trace file opens:
ply -c 'kprobe:do_sys_open { printf("opened: %s\n", mem(arg(1), "128s")); }'

# Counting vfs functions by process name:
ply -c 'kprobe:vfs_* { @[comm(), func()].count(); }'

# Counting off-CPU stacks:
ply -c 'kprobe:schedule { @[stack()].count(); }'

# Syscall read return size as a histogram:
ply -c 'kretprobe:SyS_read { @ret.quantize(retval()); }'

# Syscall read latency as a histogram:
ply -A -c 'kprobe:SyS_read { @start[tid()] = nsecs(); }
   kretprobe:SyS_read /@start[tid()]/ { @ns.quantize(nsecs() - @start[tid()]);
   @start[tid()] = nil; }'

[...]

also see ply/oneliners.md
ply

• A new BPF-based dynamic tracer for Linux
  – Created by Tobias Waldekranz

• High-level language
  – Simple one-liners
  – Short scripts

• In development
  – kprobes and tracepoints only, uprobes/perf_events not yet
  – Successful so far as a proof of concept
  – Not production tested yet (bcc is)
Future work

CHALLENGES
Challenges

• Marketing
• Documentation
• Training
• Community

Without these, we may have another ftrace: a built in "secret" of Linux. Not good for adoption! https://www.iovisor.org project helps, but tracing (observability) is only one part.
Take aways

1. Understanding the value of Linux tracing superpowers
2. Upgrade to Linux 4.4+ (4.9 is better)
3. Ask for eBPF support in your perf analysis/monitoring tools

Please contribute:
- https://github.com/iovisor/bcc
- https://github.com/iovisor/ply

BPF Tracing in Linux
- 3.19: sockets
- 3.19: maps
- 4.1: kprobes
- 4.3: uprobes
- 4.4: BPF output
- 4.6: stacks
- 4.7: tracepoints
- 4.9: profiling
- 4.9: PMCs
I’ll change your view of Linux tracing: [https://www.youtube.com/watch?v=GsMs3n8CB6g](https://www.youtube.com/watch?v=GsMs3n8CB6g)

On designing tracing tools: [https://www.youtube.com/watch?v=uibLwoVKjec](https://www.youtube.com/watch?v=uibLwoVKjec)

BPF:
- [https://github.com/iovisor/bpf-docs](https://github.com/iovisor/bpf-docs)
- [https://suchakra.wordpress.com/tag/bpf/](https://suchakra.wordpress.com/tag/bpf/)

Flame Graphs:
- [http://www.brendangregg.com/flamegraphs.html](http://www.brendangregg.com/flamegraphs.html)

Dynamic Instrumentation:
- DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD, Brendan Gregg, Jim Mauro; Prentice Hall 2011

Netflix Tech Blog on Vector:

Linux Performance: [http://www.brendangregg.com/linuxperf.html](http://www.brendangregg.com/linuxperf.html)
Thanks

- Questions?
- iovisor bcc: https://github.com/iovisor/bcc
- http://www.brendangregg.com
- http://slideshare.net/brendangregg
- bgregg@netflix.com
- @brendangregg

Thanks to Alexei Starovoitov (Facebook), Brenden Blanco (PLUMgrid/VMware), Sasha Goldshtein (Sela), Daniel Borkmann (Cisco), Wang Nan (Huawei), and other BPF and bcc contributors!