Performance Analysis
Superpowers with Linux eBPF

Brendan Gregg
Senior Performance Architect
Jun 2017

velocityconf.com
#VelocityConf
Efficiently trace TCP sessions with PID, bytes, and duration using `tcplife`

```bash
# /usr/share/bcc/tools/tcplife

<table>
<thead>
<tr>
<th>PID</th>
<th>COMM</th>
<th>LADDR</th>
<th>LPOR</th>
<th>RADDR</th>
<th>RPOR</th>
<th>TX_KB</th>
<th>RX_KB</th>
<th>MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>8078</td>
<td>100.82.130.159</td>
<td>12410</td>
<td>0</td>
<td>0</td>
<td>5.44</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>8078</td>
<td>100.82.78.215</td>
<td>55564</td>
<td>0</td>
<td>0</td>
<td>135.32</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>60778</td>
<td>100.82.207.252</td>
<td>7001</td>
<td>0</td>
<td>13</td>
<td>15126.87</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>38884</td>
<td>100.82.208.178</td>
<td>7001</td>
<td>0</td>
<td>0</td>
<td>15568.25</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>4243</td>
<td>127.0.0.1</td>
<td>42166</td>
<td>0</td>
<td>0</td>
<td>0.61</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>42166</td>
<td>127.0.0.1</td>
<td>4243</td>
<td>0</td>
<td>0</td>
<td>0.67</td>
</tr>
<tr>
<td>12030</td>
<td>upload-mes</td>
<td>127.0.0.1</td>
<td>34020</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>11</td>
<td>0</td>
<td>3.38</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>127.0.0.1</td>
<td>34020</td>
<td>0</td>
<td>11</td>
<td>3.41</td>
</tr>
<tr>
<td>12030</td>
<td>upload-mes</td>
<td>127.0.0.1</td>
<td>21196</td>
<td>127.0.0.1</td>
<td>7101</td>
<td>0</td>
<td>0</td>
<td>12.61</td>
</tr>
<tr>
<td>3964</td>
<td>mesos-slav</td>
<td>127.0.0.1</td>
<td>7101</td>
<td>127.0.0.1</td>
<td>21196</td>
<td>0</td>
<td>0</td>
<td>12.64</td>
</tr>
<tr>
<td>12021</td>
<td>upload-sys</td>
<td>127.0.0.1</td>
<td>34022</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>372</td>
<td>0</td>
<td>15.28</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>127.0.0.1</td>
<td>34022</td>
<td>0</td>
<td>372</td>
<td>15.31</td>
</tr>
<tr>
<td>2235</td>
<td>dockerd</td>
<td>100.82.34.63</td>
<td>13730</td>
<td>100.82.136.233</td>
<td>7002</td>
<td>0</td>
<td>4</td>
<td>18.50</td>
</tr>
<tr>
<td>2235</td>
<td>dockerd</td>
<td>100.82.34.63</td>
<td>34314</td>
<td>100.82.64.53</td>
<td>7002</td>
<td>0</td>
<td>8</td>
<td>56.73</td>
</tr>
<tr>
<td>12068</td>
<td>titus-reap</td>
<td>127.0.0.1</td>
<td>46476</td>
<td>127.0.0.1</td>
<td>19609</td>
<td>0</td>
<td>0</td>
<td>1.25</td>
</tr>
</tbody>
</table>
```

[...]

Enhanced BPF is in Linux
Agenda

1. eBPF & bcc
2. bcc/BPF CLI Tools
3. bcc/BPF Visualizations
Take aways

1. Identify possibilities with Linux tracing superpowers
2. Upgrade to Linux 4.4+ (4.9 is better)
3. Ask for eBPF support in your perf analysis/monitoring tools
NETFLIX

REGIONS WHERE NETFLIX IS AVAILABLE

amazon web services
ubuntu
FreeBSD
Who at Netflix will use BPF?
Introducing enhanced BPF for tracing: kernel-level software
Ye Olde BPF

Berkeley Packet Filter

```
# tcpdump host 127.0.0.1 and port 22 -d
(000) ldh [12]
(001) jeq #0x800  jt 2  jf 18
(002) ld [26]
(003) jeq #0x7f000001  jt 6  jf 4
(004) ld [30]
(005) jeq #0x7f000001  jt 6  jf 18
(006) ldb [23]
(007) jeq #0x84  jt 10  jf 8
(008) jeq #0x6  jt 10  jf 9
(009) jeq #0x11  jt 10  jf 18
(010) ldh [20]
(011) jset #0x1fff  jt 18  jf 12
(012) ldxb 4*([14]&0xf)
(013) ldh [x + 14]

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_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_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_DW, 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+
BPF for Tracing, Internals

Enhanced BPF is also now used for SDNs, DDOS mitigation, intrusion detection, container security, ...
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()
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, ...
Introducing BPF Compiler Collection: user-level software

BCC
bcc

• 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:
bcc/BPF (C & Python)

```python
# 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;
}"")
```

```c
# 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

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

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

entire program
The Tracing Landscape, Jun 2017

Ease of use

(more brutal) |
---
(brutal) | (alpha) | (mature) |
---
Stage of Development

Scope & Capability

(my opinion)

dtrace4L. |
---
ply/BPF |
---
ktap |
---
stap |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
---
 perf |
Performance analysis

BCC/BPF CLI TOOLS
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:
  ```
  # echo "deb [trusted=yes] https://repo.iovisor.org/apt/xenial xenial-nightly main" \ 
  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
Discover short-lived process issues using *execsnoop*

```bash
# execsnoop -t

<table>
<thead>
<tr>
<th>TIME(s)</th>
<th>PCOMM</th>
<th>PID</th>
<th>PPID</th>
<th>RET</th>
<th>ARGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.031</td>
<td>dirname</td>
<td>23832</td>
<td>23808</td>
<td>0</td>
<td>/usr/bin/dirname /apps/tomcat/bin/catalina.sh</td>
</tr>
<tr>
<td>0.888</td>
<td>run</td>
<td>23833</td>
<td>2344</td>
<td>0</td>
<td>./run</td>
</tr>
<tr>
<td>0.889</td>
<td>run</td>
<td>23833</td>
<td>2344</td>
<td>-2</td>
<td>/command/bash</td>
</tr>
<tr>
<td>0.889</td>
<td>run</td>
<td>23833</td>
<td>2344</td>
<td>-2</td>
<td>/usr/local/bin/bash</td>
</tr>
<tr>
<td>0.889</td>
<td>run</td>
<td>23833</td>
<td>2344</td>
<td>-2</td>
<td>/usr/local/sbin/bash</td>
</tr>
<tr>
<td>0.889</td>
<td>bash</td>
<td>23833</td>
<td>2344</td>
<td>0</td>
<td>/bin/bash</td>
</tr>
<tr>
<td>0.894</td>
<td>svstat</td>
<td>23835</td>
<td>23834</td>
<td>0</td>
<td>/command/svstat /service/nflx-httpd</td>
</tr>
<tr>
<td>0.894</td>
<td>perl</td>
<td>23836</td>
<td>23834</td>
<td>0</td>
<td>/usr/bin/perl --e $l=&lt;&gt;;$l~/(\d+) sec/;print $1</td>
</tr>
<tr>
<td>0.899</td>
<td>ps</td>
<td>23838</td>
<td>23837</td>
<td>0</td>
<td>/bin/ps --ppid 1 -o pid,cmd,args</td>
</tr>
<tr>
<td>0.900</td>
<td>grep</td>
<td>23839</td>
<td>23837</td>
<td>0</td>
<td>/bin/grep org.apache.catalina</td>
</tr>
<tr>
<td>0.900</td>
<td>sed</td>
<td>23840</td>
<td>23837</td>
<td>0</td>
<td>/bin/sed s/^ *///</td>
</tr>
<tr>
<td>0.900</td>
<td>cut</td>
<td>23841</td>
<td>23837</td>
<td>0</td>
<td>/usr/bin/cut -d -f 1</td>
</tr>
<tr>
<td>0.901</td>
<td>xargs</td>
<td>23842</td>
<td>23837</td>
<td>0</td>
<td>/usr/bin/xargs</td>
</tr>
<tr>
<td>0.912</td>
<td>xargs</td>
<td>23843</td>
<td>23842</td>
<td>-2</td>
<td>/command/echo</td>
</tr>
<tr>
<td>0.912</td>
<td>xargs</td>
<td>23843</td>
<td>23842</td>
<td>-2</td>
<td>/usr/local/bin/echo</td>
</tr>
<tr>
<td>0.912</td>
<td>xargs</td>
<td>23843</td>
<td>23842</td>
<td>-2</td>
<td>/usr/local/sbin/echo</td>
</tr>
<tr>
<td>0.912</td>
<td>echo</td>
<td>23843</td>
<td>23842</td>
<td>0</td>
<td>/bin/echo</td>
</tr>
</tbody>
</table>

[...]
Discover short-lived process issues using execsnoop

```plaintext
# execsnoop -t

<table>
<thead>
<tr>
<th>TIME(s)</th>
<th>PCOMM</th>
<th>PID</th>
<th>PPID</th>
<th>RET</th>
<th>ARGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.031</td>
<td>dirname</td>
<td>23832</td>
<td>23808</td>
<td>0</td>
<td>/usr/bin/dirmame /apps/tomcat/bin/catalina.sh</td>
</tr>
<tr>
<td>0.888</td>
<td>run</td>
<td>23833</td>
<td>2344</td>
<td>-2</td>
<td>/command/bash</td>
</tr>
<tr>
<td>0.888</td>
<td>run</td>
<td>23833</td>
<td>2344</td>
<td>-2</td>
<td>/usr/local/bin/bash</td>
</tr>
<tr>
<td>0.889</td>
<td>run</td>
<td>23833</td>
<td>2344</td>
<td>-2</td>
<td>/usr/local/sbin/bash</td>
</tr>
<tr>
<td>0.889</td>
<td>bash</td>
<td>23833</td>
<td>2344</td>
<td>0</td>
<td>/bin/bash</td>
</tr>
<tr>
<td>0.894</td>
<td>svstat</td>
<td>23835</td>
<td>23834</td>
<td>0</td>
<td>/command/svstat /service/nflx-httpd</td>
</tr>
<tr>
<td>0.894</td>
<td>perl</td>
<td>23836</td>
<td>23834</td>
<td>0</td>
<td>/usr/bin/perl -e $l=&lt;$; $l=~/(\d+) sec/; print $1</td>
</tr>
<tr>
<td>0.899</td>
<td>ps</td>
<td>23838</td>
<td>23837</td>
<td>0</td>
<td>/bin/ps --ppid 1 -o pid,cmd,args</td>
</tr>
<tr>
<td>0.900</td>
<td>grep</td>
<td>23839</td>
<td>23837</td>
<td>0</td>
<td>/bin/grep org.apache.catalina</td>
</tr>
<tr>
<td>0.900</td>
<td>sed</td>
<td>23840</td>
<td>23837</td>
<td>0</td>
<td>/bin/sed s/^ */;</td>
</tr>
<tr>
<td>0.901</td>
<td>cut</td>
<td>23841</td>
<td>23837</td>
<td>0</td>
<td>/usr/bin/cut -d -f 1</td>
</tr>
<tr>
<td>0.912</td>
<td>xargs</td>
<td>23842</td>
<td>23837</td>
<td>0</td>
<td>/usr/bin/xargs</td>
</tr>
<tr>
<td>0.912</td>
<td>xargs</td>
<td>23843</td>
<td>23842</td>
<td>-2</td>
<td>/command/echo</td>
</tr>
<tr>
<td>0.912</td>
<td>xargs</td>
<td>23843</td>
<td>23842</td>
<td>-2</td>
<td>/usr/local/bin/echo</td>
</tr>
<tr>
<td>0.912</td>
<td>echo</td>
<td>23843</td>
<td>23842</td>
<td>-2</td>
<td>/usr/local/sbin/echo</td>
</tr>
</tbody>
</table>
```

[...]
Exonerate or confirm storage latency issues and outliers with `ext4slower`

```bash
# /usr/share/bcc/tools/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>17:31:42</td>
<td>postdrop</td>
<td>15523</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>2.32</td>
<td>5630D406E4</td>
</tr>
<tr>
<td>17:31:42</td>
<td>cleanup</td>
<td>15524</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>1.89</td>
<td>57BB7406EC</td>
</tr>
<tr>
<td>17:32:09</td>
<td>titus-log-ship</td>
<td>19735</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>1.94</td>
<td>slurper_checkpoint.db</td>
</tr>
<tr>
<td>17:35:37</td>
<td>dhclient</td>
<td>1061</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>3.32</td>
<td>dhclient.eth0.leases</td>
</tr>
<tr>
<td>17:35:39</td>
<td>systemd-journ</td>
<td>504</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>26.62</td>
<td>system.journal</td>
</tr>
<tr>
<td>17:35:39</td>
<td>systemd-journ</td>
<td>504</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>1.56</td>
<td>system.journal</td>
</tr>
<tr>
<td>17:35:39</td>
<td>systemd-journ</td>
<td>504</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>1.73</td>
<td>system.journal</td>
</tr>
<tr>
<td>17:35:45</td>
<td>postdrop</td>
<td>16187</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>2.41</td>
<td>C0369406E4</td>
</tr>
<tr>
<td>17:35:45</td>
<td>cleanup</td>
<td>16188</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>6.52</td>
<td>C1B90406EC</td>
</tr>
</tbody>
</table>
```

Tracing at the file system is a more reliable and complete indicator than measuring disk I/O latency

Also: `btrfslower`, `xfsslower`, `zfsslower`
Exonerate or confirm storage latency issues and outliers with `ext4slower`

```
# /usr/share/bcc/tools/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>17:31:42</td>
<td>postdrop</td>
<td>15523</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>2.32</td>
<td>5630D406E4</td>
</tr>
<tr>
<td>17:31:42</td>
<td>cleanup</td>
<td>15524</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>1.89</td>
<td>57BB7406EC</td>
</tr>
<tr>
<td>17:32:09</td>
<td>titus-log-ship</td>
<td>19735</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>1.94</td>
<td>slurper_checkpoint.db</td>
</tr>
<tr>
<td>17:35:37</td>
<td>dhclient</td>
<td>1061</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>3.32</td>
<td>dhclient.eth0.leases</td>
</tr>
<tr>
<td>17:35:39</td>
<td>systemd-journa</td>
<td>504</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>26.62</td>
<td>system.journal</td>
</tr>
<tr>
<td>17:35:39</td>
<td>systemd-journa</td>
<td>504</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>1.56</td>
<td>system.journal</td>
</tr>
<tr>
<td>17:35:39</td>
<td>systemd-journa</td>
<td>504</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>1.73</td>
<td>system.journal</td>
</tr>
<tr>
<td>17:35:45</td>
<td>postdrop</td>
<td>16187</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>2.41</td>
<td>C0369406E4</td>
</tr>
<tr>
<td>17:35:45</td>
<td>cleanup</td>
<td>16188</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>6.52</td>
<td>C1B90406EC</td>
</tr>
</tbody>
</table>

[...]```

Tracing at the file system is a more reliable and complete indicator than measuring disk I/O latency

Also: `btrfs slower`, `xfsslower`, `zfsslower`
Identify multimodal disk I/O latency and outliers with biolatency

```
# biolatency -mT 10
Tracing block device I/O... Hit Ctrl-C to end.

19:19:04

<table>
<thead>
<tr>
<th>msecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>238</td>
<td>***********</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>424</td>
<td>*******************</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>834</td>
<td>***********************</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>506</td>
<td>***********************</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>986</td>
<td>****************************</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>97</td>
<td>***</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>27</td>
<td>*</td>
</tr>
</tbody>
</table>

19:19:14

<table>
<thead>
<tr>
<th>msecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>427</td>
<td>*******************</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>424</td>
<td>*******************</td>
</tr>
</tbody>
</table>

[...]
```

Average latency (iostat/sar) may not be representitive with multiple modes or outliers
Identify multimodal disk I/O latency and outliers with `biolatency`

```bash
# biolatency -mT 10
Tracing block device I/O... Hit Ctrl-C to end.
```

19:19:04

<table>
<thead>
<tr>
<th>mssecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>238</td>
<td>**********</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>424</td>
<td>****************</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>834</td>
<td>*******************</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>506</td>
<td>****************</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>986</td>
<td>*******************</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>97</td>
<td>***</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>27</td>
<td>*</td>
</tr>
</tbody>
</table>

19:19:14

<table>
<thead>
<tr>
<th>mssecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>427</td>
<td>****************</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>424</td>
<td>****************</td>
</tr>
</tbody>
</table>

[...]```

Average latency (iostat/sar) may not be representitive with multiple modes or outliers
Efficiently trace TCP sessions with PID, bytes, and duration using `tcplife`

```
# /usr/share/bcc/tools/tcplife

<table>
<thead>
<tr>
<th>PID</th>
<th>COMM</th>
<th>LADDR</th>
<th>LPORT</th>
<th>RADDR</th>
<th>RPORT</th>
<th>TX_KB</th>
<th>RX_KB</th>
<th>MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>8078</td>
<td>100.82.130.159</td>
<td>12410</td>
<td>0</td>
<td>0</td>
<td>5.44</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>8078</td>
<td>100.82.78.215</td>
<td>55564</td>
<td>0</td>
<td>0</td>
<td>135.32</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>60778</td>
<td>100.82.207.252</td>
<td>7001</td>
<td>0</td>
<td>13</td>
<td>15126.87</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>38884</td>
<td>100.82.208.178</td>
<td>7001</td>
<td>0</td>
<td>0</td>
<td>15568.25</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>4243</td>
<td>127.0.0.1</td>
<td>42166</td>
<td>0</td>
<td>0</td>
<td>0.61</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>42166</td>
<td>127.0.0.1</td>
<td>4243</td>
<td>0</td>
<td>0</td>
<td>0.67</td>
</tr>
<tr>
<td>12030</td>
<td>upload-mes</td>
<td>127.0.0.1</td>
<td>34020</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>11</td>
<td>0</td>
<td>3.38</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>127.0.0.1</td>
<td>34020</td>
<td>0</td>
<td>11</td>
<td>3.41</td>
</tr>
<tr>
<td>12030</td>
<td>upload-mes</td>
<td>127.0.0.1</td>
<td>21196</td>
<td>127.0.0.1</td>
<td>7101</td>
<td>0</td>
<td>0</td>
<td>12.61</td>
</tr>
<tr>
<td>3964</td>
<td>mesos-slav</td>
<td>127.0.0.1</td>
<td>7101</td>
<td>127.0.0.1</td>
<td>21196</td>
<td>0</td>
<td>0</td>
<td>12.64</td>
</tr>
<tr>
<td>12021</td>
<td>upload-sys</td>
<td>127.0.0.1</td>
<td>34022</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>372</td>
<td>0</td>
<td>15.28</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>127.0.0.1</td>
<td>34022</td>
<td>0</td>
<td>372</td>
<td>15.31</td>
</tr>
<tr>
<td>2235</td>
<td>dockerd</td>
<td>100.82.34.63</td>
<td>13730</td>
<td>100.82.136.233</td>
<td>7002</td>
<td>0</td>
<td>4</td>
<td>18.50</td>
</tr>
<tr>
<td>2235</td>
<td>dockerd</td>
<td>100.82.34.63</td>
<td>34314</td>
<td>100.82.64.53</td>
<td>7002</td>
<td>0</td>
<td>8</td>
<td>56.73</td>
</tr>
</tbody>
</table>

[...]
```

Dynamic tracing of TCP set state only; does not trace send/receive
Also see: tcpconnect, tcpaccept, tcpretrans
### Efficiently trace TCP sessions with PID, bytes, and duration using `tcplife`

```plaintext
# /usr/share/bcc/tools/tcplife

<table>
<thead>
<tr>
<th>PID</th>
<th>COMM</th>
<th>LADDR</th>
<th>LPOR T</th>
<th>RADDR</th>
<th>RPORT</th>
<th>TX_KB</th>
<th>RX_KB</th>
<th>MS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>8078</td>
<td>100.82.130.159</td>
<td>12410</td>
<td>0</td>
<td>0</td>
<td>5.44</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>8078</td>
<td>100.82.78.215</td>
<td>55564</td>
<td>0</td>
<td>0</td>
<td>135.32</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>60778</td>
<td>100.82.207.252</td>
<td>7001</td>
<td>13</td>
<td>15126.87</td>
<td></td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>100.82.34.63</td>
<td>38884</td>
<td>100.82.208.178</td>
<td>7001</td>
<td>0</td>
<td>0</td>
<td>15568.25</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>4243</td>
<td>127.0.0.1</td>
<td>42166</td>
<td>0</td>
<td>0</td>
<td>0.61</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>42166</td>
<td>127.0.0.1</td>
<td>4243</td>
<td>0</td>
<td>0</td>
<td>0.67</td>
</tr>
<tr>
<td>12030</td>
<td>upload-mes</td>
<td>127.0.0.1</td>
<td>34020</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>11</td>
<td>0</td>
<td>3.38</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>127.0.0.1</td>
<td>34020</td>
<td>0</td>
<td>11</td>
<td>3.41</td>
</tr>
<tr>
<td>12030</td>
<td>upload-mes</td>
<td>127.0.0.1</td>
<td>21196</td>
<td>127.0.0.1</td>
<td>7101</td>
<td>0</td>
<td>0</td>
<td>12.61</td>
</tr>
<tr>
<td>3964</td>
<td>mesos-slav</td>
<td>127.0.0.1</td>
<td>7101</td>
<td>127.0.0.1</td>
<td>21196</td>
<td>0</td>
<td>0</td>
<td>12.64</td>
</tr>
<tr>
<td>12021</td>
<td>upload-sys</td>
<td>127.0.0.1</td>
<td>34022</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>372</td>
<td>0</td>
<td>15.28</td>
</tr>
<tr>
<td>2509</td>
<td>java</td>
<td>127.0.0.1</td>
<td>8078</td>
<td>127.0.0.1</td>
<td>34022</td>
<td>0</td>
<td>372</td>
<td>15.31</td>
</tr>
<tr>
<td>2235</td>
<td>dockerd</td>
<td>100.82.34.63</td>
<td>13730</td>
<td>100.82.136.233</td>
<td>7002</td>
<td>0</td>
<td>4</td>
<td>18.50</td>
</tr>
<tr>
<td>2235</td>
<td>dockerd</td>
<td>100.82.34.63</td>
<td>34314</td>
<td>100.82.64.53</td>
<td>7002</td>
<td>0</td>
<td>8</td>
<td>56.73</td>
</tr>
</tbody>
</table>

[...]
```

Dynamic tracing of TCP set state only; does *not* trace send/receive
Also see: tcpconnect, tcpaccept, tcpretrans
### Identify DNS latency issues system wide with gethostlatency

Instruments using user-level dynamic tracing of getaddrinfo(), gethostbyname(), etc.
Identify DNS latency issues system wide with `gethostlatency`

```bash
# /usr/share/bcc/tools/gethostlatency

<table>
<thead>
<tr>
<th>TIME</th>
<th>PID</th>
<th>COMM</th>
<th>LATms</th>
<th>HOST</th>
</tr>
</thead>
<tbody>
<tr>
<td>18:56:36</td>
<td>5055</td>
<td>mesos-slave</td>
<td>0.01</td>
<td>100.82.166.217</td>
</tr>
<tr>
<td>18:56:40</td>
<td>5590</td>
<td>java</td>
<td>3.53</td>
<td>ec2-....-79.compute-1.amazonaws.com</td>
</tr>
<tr>
<td>18:56:51</td>
<td>5055</td>
<td>mesos-slave</td>
<td>0.01</td>
<td>100.82.166.217</td>
</tr>
<tr>
<td>18:56:53</td>
<td>30166</td>
<td>ncat</td>
<td>0.21</td>
<td>localhost</td>
</tr>
<tr>
<td>18:56:56</td>
<td>6661</td>
<td>java</td>
<td>2.19</td>
<td>atlas-alert-.....prod.netflix.net</td>
</tr>
<tr>
<td>18:56:59</td>
<td>5589</td>
<td>java</td>
<td>1.50</td>
<td>ec2-....-207.compute-1.amazonaws.com</td>
</tr>
<tr>
<td>18:57:03</td>
<td>5370</td>
<td>java</td>
<td>0.04</td>
<td>localhost</td>
</tr>
<tr>
<td>18:57:03</td>
<td>30259</td>
<td>sudo</td>
<td>0.07</td>
<td>titusagent-mainvpc-m...3465</td>
</tr>
<tr>
<td>18:57:06</td>
<td>5055</td>
<td>mesos-slave</td>
<td>0.01</td>
<td>100.82.166.217</td>
</tr>
<tr>
<td>18:57:10</td>
<td>5590</td>
<td>java</td>
<td>3.10</td>
<td>ec2-....-79.compute-1.amazonaws.com</td>
</tr>
<tr>
<td>18:57:21</td>
<td>5055</td>
<td>mesos-slave</td>
<td>0.01</td>
<td>100.82.166.217</td>
</tr>
<tr>
<td>18:57:29</td>
<td>5589</td>
<td>java</td>
<td>52.36</td>
<td>ec2-....-207.compute-1.amazonaws.com</td>
</tr>
<tr>
<td>18:57:36</td>
<td>5055</td>
<td>mesos-slave</td>
<td>0.01</td>
<td>100.82.166.217</td>
</tr>
<tr>
<td>18:57:40</td>
<td>5590</td>
<td>java</td>
<td>1.83</td>
<td>ec2-....-79.compute-1.amazonaws.com</td>
</tr>
<tr>
<td>18:57:51</td>
<td>5055</td>
<td>mesos-slave</td>
<td>0.01</td>
<td>100.82.166.217</td>
</tr>
</tbody>
</table>

[...]
```

Instruments using user-level dynamic tracing of `getaddrinfo()`, `gethostbyname()`, etc.
Examine CPU scheduler run queue latency as a histogram with runqlat

# /usr/share/bcc/tools/runqlat 10
Tracing run queue latency... Hit Ctrl-C to end.

<table>
<thead>
<tr>
<th>usecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>2810</td>
<td>*</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>5248</td>
<td>**</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>12369</td>
<td>******</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>71312</td>
<td>****************************************</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>55705</td>
<td>*******************</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>11775</td>
<td>*****</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>6230</td>
<td>***</td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>2758</td>
<td>*</td>
</tr>
<tr>
<td>256 -&gt; 511</td>
<td>549</td>
<td></td>
</tr>
<tr>
<td>512 -&gt; 1023</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>1024 -&gt; 2047</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>2048 -&gt; 4095</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4096 -&gt; 8191</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

[...]

As efficient as possible: scheduler calls can become frequent
Examine CPU scheduler run queue latency as a histogram with `runqlat`

```
# /usr/share/bcc/tools/runqlat 10
Tracing run queue latency... Hit Ctrl-C to end.

<table>
<thead>
<tr>
<th>usecs</th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 -&gt; 1</td>
<td>2810</td>
<td>*</td>
</tr>
<tr>
<td>2 -&gt; 3</td>
<td>5248</td>
<td>**</td>
</tr>
<tr>
<td>4 -&gt; 7</td>
<td>12369</td>
<td>*******</td>
</tr>
<tr>
<td>8 -&gt; 15</td>
<td>71312</td>
<td>******************</td>
</tr>
<tr>
<td>16 -&gt; 31</td>
<td>55705</td>
<td>****************************************</td>
</tr>
<tr>
<td>32 -&gt; 63</td>
<td>11775</td>
<td>*******</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>6230</td>
<td>***</td>
</tr>
<tr>
<td>128 -&gt; 255</td>
<td>2758</td>
<td>*</td>
</tr>
<tr>
<td>256 -&gt; 511</td>
<td>549</td>
<td></td>
</tr>
<tr>
<td>512 -&gt; 1023</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>1024 -&gt; 2047</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>2048 -&gt; 4095</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4096 -&gt; 8191</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
```

As efficient as possible: scheduler calls can become frequent
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
- Start with the Q's, then find the A's
- Use multi-tools:
  - funccount, trace, argdist, stackcount

E.g., storage I/O subsystem:
Construct programmatic one-liners with trace

e.g. reads over 20000 bytes:

```bash
# 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
[...]
Create in-kernel summaries with \texttt{argdist}

e.g. histogram of \texttt{tcp\_cleanup\_rbuf()} copied:

\begin{verbatim}
# argdist -H 'p::tcp_cleanup_rbuf(struct sock *sk, int copied):int:copied'
[15:34:45]

copied : count distribution
   0 -> 1 : 15088 |******************************************|
   2 -> 3 : 0
   4 -> 7 : 0
   8 -> 15: 0
  16 -> 31: 0
  32 -> 63: 0
  64 -> 127: 4786 |***********|
 128 -> 255: 1
 256 -> 511: 1
 512 -> 1023: 4
1024 -> 2047: 11
2048 -> 4095: 5
4096 -> 8191: 27
 8192 -> 16383: 105
16384 -> 32767: 0
\end{verbatim}

argdist by Sasha Goldshtein
Coming to a GUI near you

BCC/BPF VISUALIZATIONS
BPF metrics and analysis can be automated in GUIs

Eg, Netflix Vector (self-service UI):

Should be open sourced; you may also build/buy your own
Latency heatmaps show histograms over time
CPU and Off-CPU Flame Graphs can be BPF optimized and used in production

http://www.brendangregg.com/flamegraphs.html
On-CPU with Off-CPU analysis can help solve any performance issue

Generic thread state transition diagram
Advanced off-CPU analysis: BPF can merge the blocking stack with the waker stack in-kernel
FUTURE WORK

bcc/BPF
Challenges

• Adoption
  – Performance monitoring/analysis products
• Marketing
• Documentation
• Training
• Community
• Higher-level language
ply

- A new BPF-based language and tracer for Linux
  - Created by Tobias Waldekranz
  - [https://github.com/iovisor/ply](https://github.com/iovisor/ply)  [https://wkz.github.io/ply/](https://wkz.github.io/ply/)

- 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)
File opens can be traced using a short ply one-liner

# 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: .
[...]
ply programs are concise, such as measuring 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

[...] @ns:
```

```
|   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 |
```
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
Links & References

iovisor bcc:
- http://www.brendangregg.com/blog/ (search for "bcc")
- http://www.brendangregg.com/ebpf.html#bcc
- On designing tracing tools: https://www.youtube.com/watch?v=uibLwoVKjec

bcc tutorial:
- https://github.com/iovisor/bcc/blob/master/INSTALL.md
- .../docs/tutorial.md
- .../docs/tutorial_bcc_python_developer.md
- .../docs/reference_guide.md
- .../CONTRIBUTING-SCRIPTS.md

ply: https://github.com/iovisor/ply

BPF:
- https://github.com/iovisor/bpf-docs
- https://suchakra.wordpress.com/tag/bpf/

Flame Graphs:
- http://www.brendangregg.com/flamegraphs.html

Netflix Tech Blog on Vector:

Linux Performance: http://www.brendangregg.com/linuxperf.html
Thank You

- 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!