Performance Analysis Superpowers with Linux eBPF

Brendan Gregg
Senior Performance Architect
Jul 2017
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>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>8078</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>
```

[...]
Agenda

1. eBPF & bcc

2. bcc/BPF CLI Tools

3. bcc/BPF Visualizations
Take aways

1. Understand Linux tracing components
2. Understand the role and state of enhanced BPF
3. Discover opportunities for future development
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

<table>
<thead>
<tr>
<th>Line</th>
<th>Instruction</th>
<th>Value</th>
<th>Address</th>
<th>Condition</th>
<th>Jump to</th>
<th>Jump from</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>ldh</td>
<td>[12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>001</td>
<td>jeq</td>
<td>#0x800</td>
<td></td>
<td></td>
<td>jt 2</td>
<td>jf 18</td>
</tr>
<tr>
<td>002</td>
<td>ld</td>
<td>[26]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>003</td>
<td>jeq</td>
<td>#0x7f000001</td>
<td></td>
<td></td>
<td>jt 6</td>
<td>jf 4</td>
</tr>
<tr>
<td>004</td>
<td>ld</td>
<td>[30]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>005</td>
<td>jeq</td>
<td>#0x7f000001</td>
<td></td>
<td></td>
<td>jt 6</td>
<td>jf 18</td>
</tr>
<tr>
<td>006</td>
<td>ldb</td>
<td>[23]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>007</td>
<td>jeq</td>
<td>#0x84</td>
<td></td>
<td></td>
<td>jt 10</td>
<td>jf 8</td>
</tr>
<tr>
<td>008</td>
<td>jeq</td>
<td>#0x6</td>
<td></td>
<td></td>
<td>jt 10</td>
<td>jf 9</td>
</tr>
<tr>
<td>009</td>
<td>jeq</td>
<td>#0x11</td>
<td></td>
<td></td>
<td>jt 10</td>
<td>jf 18</td>
</tr>
<tr>
<td>010</td>
<td>ldh</td>
<td>[20]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>jset</td>
<td>#0x1fff</td>
<td></td>
<td></td>
<td>jt 18</td>
<td>jf 12</td>
</tr>
<tr>
<td>012</td>
<td>ldxb</td>
<td>4*([14]&amp;0xf)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>013</td>
<td>ldh</td>
<td>[x + 14]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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, …
Dynamic Tracing

Dynamic Program Instrumentation for Scalable Performance Tools

Jeffrey K. Hollingsworth
hollings@cs.wisc.edu

Barton P. Miller
bart@cs.wisc.edu

Jon Cargille
jon@cs.wisc.edu

Computer Sciences Department
University of Wisconsin-Madison

Abstract

In this paper, we present a new technique called dynamic instrumentation that provides efficient, scalable, yet detailed data collection for large-scale parallel applications. Our approach is unique because it defers inserting any instrumentation until the application is in execution. We can insert or change instrumentation at any time during execution by modifying the application’s binary image. Only the instrumentation required for the currently selected analysis or visualization is inserted. As a result, our technique collects several orders of magnitude less data than traditional data collection approaches. We have implemented a prototype of our dynamic instrumentation on the CM-5, and present results understand the bottlenecks in their program. It must be frugal so that the instrumentation overhead does not obscure or distort the bottlenecks in the original program. The instrumentation system must also scale to large, production data set sizes and number of processors.

A detailed instrumentation system needs to be able to collect data about each component of a parallel machine. To correct bottlenecks, programmers need to know as precisely as possible how the utilization of these components is hindering the performance of their program.

There are two ways to provide frugal instrumentation: make data collection efficient, or collect less data. All tool builders strive to make their data collection more
1999: Kerninst

http://www.paradyn.org/html/kerninst.html
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()
Linux Events & BPF Support

BPF output
- Linux 4.4

BPF stacks
- Linux 4.6

Dynamic Tracing
- uprobes
  - Linux 4.3
- kprobes
  - Linux 4.1

(version
BPF support arrived)

Tracepoints
- Linux 4.7
  - ext4:
  - sock:
  - sched:
  - signal:
  - timer:
  - workqueue:

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

PMCs
- Linux 4.9
  - cycles instructions
  - branch- *
  - L1- *
  - LLC- *

CPU INTERCONNECT

VFS

System Call Interface

Applications

System Libraries

Device Drivers

Device Interf\ce
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 Complier Collection: user-level front-end

BCC
bcc

- BPF Compiler Collection
  - [https://github.com/iovisor/bcc](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_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(),
};
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 = _(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

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

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

entire program
The Tracing Landscape, Jul 2017

Ease of use
(brutal) → (mature)

Stage of Development

(my opinion)

recent changes

(hist triggers)

(dtrace4L.)

ply/BPF

ktap

perf

stap

LTTng

ftrace

bcc/BPF

C/BPF

Raw BPF
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:

```bash
# 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

```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/dircme /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-htpd</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>

[...]  

Efficient: only traces exec()
Discover short-lived process issues using `execsnoop`

<table>
<thead>
<tr>
<th># execsnoop -t</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TIME(s)</strong></td>
</tr>
<tr>
<td>0.031</td>
</tr>
<tr>
<td>0.888</td>
</tr>
<tr>
<td>0.889</td>
</tr>
<tr>
<td>0.889</td>
</tr>
<tr>
<td>0.889</td>
</tr>
<tr>
<td>0.889</td>
</tr>
<tr>
<td>0.894</td>
</tr>
<tr>
<td>0.894</td>
</tr>
<tr>
<td>0.899</td>
</tr>
<tr>
<td>0.900</td>
</tr>
<tr>
<td>0.900</td>
</tr>
<tr>
<td>0.900</td>
</tr>
<tr>
<td>0.901</td>
</tr>
<tr>
<td>0.912</td>
</tr>
<tr>
<td>0.912</td>
</tr>
<tr>
<td>0.912</td>
</tr>
<tr>
<td>0.912</td>
</tr>
</tbody>
</table>

Efficient: only traces exec()
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-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: `btrfslower`, `xfsslower`, `zfsslower`
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`
# biolatency -mT 10
Tracing block device I/O... Hit Ctrl-C to end.

The "count" column is summarized in-kernel

<table>
<thead>
<tr>
<th>19:19:04</th>
<th></th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 -&gt; 1</td>
<td>238</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 -&gt; 3</td>
<td>424</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 -&gt; 7</td>
<td>834</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 -&gt; 15</td>
<td>506</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16 -&gt; 31</td>
<td>986</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32 -&gt; 63</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64 -&gt; 127</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128 -&gt; 255</td>
<td>27</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>19:19:14</th>
<th></th>
<th>count</th>
<th>distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0 -&gt; 1</td>
<td>427</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 -&gt; 3</td>
<td>424</td>
</tr>
</tbody>
</table>

[...]

Average latency (iostat/sar) may not be representative 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>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>

[...]
```

The "count" column is summarized in-kernel.

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

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

<table>
<thead>
<tr>
<th>PID</th>
<th>COMM</th>
<th>LADDR</th>
<th>LPOR</th>
<th>RADDR</th>
<th>RPO</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
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.
Identify DNS latency issues system wide with `gethostlatency`

```
# /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.

usecs     : count          distribution
0 -> 1    : 2810          |
2 -> 3    : 5248          |
4 -> 7    : 12369         |
8 -> 15   : 71312         |
16 -> 31  : 55705         |
32 -> 63  : 11775         |
64 -> 127 : 6230          |
128 -> 255: 2758          |
256 -> 511: 549           |
512 -> 1023: 46           |
1024 -> 2047: 11          |
2048 -> 4095: 4           |
4096 -> 8191: 5           |
```

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

```sh
# /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 - 1</td>
<td>2810</td>
<td>*</td>
</tr>
<tr>
<td>2 - 3</td>
<td>5248</td>
<td>**</td>
</tr>
<tr>
<td>4 - 7</td>
<td>12369</td>
<td>*****</td>
</tr>
<tr>
<td>8 - 15</td>
<td>71312</td>
<td>******************************</td>
</tr>
<tr>
<td>16 - 31</td>
<td>55705</td>
<td>****************************************</td>
</tr>
<tr>
<td>32 - 63</td>
<td>11775</td>
<td>*****</td>
</tr>
<tr>
<td>64 - 127</td>
<td>6230</td>
<td>***</td>
</tr>
<tr>
<td>128 - 255</td>
<td>2758</td>
<td>*</td>
</tr>
<tr>
<td>256 - 511</td>
<td>549</td>
<td></td>
</tr>
<tr>
<td>512 - 1023</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>1024 - 2047</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>2048 - 4095</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>4096 - 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'
```

<table>
<thead>
<tr>
<th>TIME</th>
<th>PID</th>
<th>COMM</th>
<th>FUNC</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>05:18:23</td>
<td>4490</td>
<td>dd</td>
<td>sys_read</td>
<td>read 1048576 bytes</td>
</tr>
<tr>
<td>05:18:23</td>
<td>4490</td>
<td>dd</td>
<td>sys_read</td>
<td>read 1048576 bytes</td>
</tr>
<tr>
<td>05:18:23</td>
<td>4490</td>
<td>dd</td>
<td>sys_read</td>
<td>read 1048576 bytes</td>
</tr>
</tbody>
</table>

^C

```bash
# 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
[...]
```

argdist by Sasha Goldshtein
Create in-kernel summaries with argdist

e.g. histogram of tcp_cleanup_rbuf() copied:

```bash
$ argdist -H 'p::tcp_cleanup_rbuf(struct sock *sk, int copied):int:copied'

<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>4786</td>
<td>***********</td>
</tr>
<tr>
<td>64 -&gt; 127</td>
<td>4786</td>
<td></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>
```

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
Efficient CPU and off-CPU flame graphs by counting stacks in kernel context
Advanced off-CPU analysis: BPF can merge the blocking stack with the waker stack in-kernel

Diagram showing the stack direction with waker task, waker stack, wokeup, blocked stack, and blocked task.
BPF

FUTURE WORK
BCC Improvements

Challenges:

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

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

// pull in details
u16 family = 0, lport = 0, dport = 0;
char state = 0;
bpf_probe_read(&family, sizeof(family), &skp->__sk_common__

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));
```
Higher-level Language

• bcc's Python/C interface is ok, but verbose
• Alternate higher-level language front end?
  – New front-ends can use existing libbcc, and can be added as part of bcc itself
  – Whave a problem in search of a new language (instead of the other way around)
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:

<table>
<thead>
<tr>
<th>Size</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>[512, 1k)</td>
<td>3</td>
</tr>
<tr>
<td>[1k, 2k)</td>
<td>7</td>
</tr>
<tr>
<td>[2k, 4k)</td>
<td>12</td>
</tr>
<tr>
<td>[4k, 8k)</td>
<td>3</td>
</tr>
<tr>
<td>[8k, 16k)</td>
<td>2</td>
</tr>
<tr>
<td>[16k, 32k)</td>
<td>0</td>
</tr>
<tr>
<td>[32k, 64k)</td>
<td>0</td>
</tr>
<tr>
<td>[64k, 128k)</td>
<td>3</td>
</tr>
<tr>
<td>[128k, 256k)</td>
<td>1</td>
</tr>
<tr>
<td>[256k, 512k)</td>
<td>1</td>
</tr>
<tr>
<td>[512k, 1M)]</td>
<td>2</td>
</tr>
</tbody>
</table>
New Tooling/Metrics

```
filetop
cachestat
trace
mdflush
btrfsdist
biotop

filelifefilesloower
 cachetop dcstact dcsnoop mountsnoop

trace argdist funccount funcslower funclatency stackcount profile

filehier filesnooper vfscount vfsstat

opensnoopy statsnoopy syncsnoopy

ucalls uflow ugc uobjnew ustat uthreads

filetop
filelifefilesloower
vfscount vfsstat

opoensnoopy statsnoopy syncsnoopy

c* java* node* php* python* ruby*

mysql_qslower bashreadline

gethostlatency memleak sslniff

Other:
capable

syscount killsnook
execsnoopy pidpersec
cpudist runqlat runqlen
deadlock_detector cpuunclaimed

offcputime wakeuptime offwaketime
softirqs
oomkill memleak slabratetop

llcstat

CPU

DRAM

https://github.com/iovisor/bcc#tools 2017
```
New Visualizations
Case Studies

• Use it
• Solve something
• Write about it
• Talk about it
Take aways

1. Understand Linux tracing components
2. Understand the role and state of enhanced BPF
3. Discover opportunities for future development

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

**bcc tutorial:**
- [https://github.com/iovisor/bcc/blob/master/INSTALL.md](https://github.com/iovisor/bcc/blob/master/INSTALL.md)
- […/docs/tutorial.md](https://github.com/iovisor/bcc/blob/master/…/docs/tutorial.md)
- […/docs/tutorial_bcc_python_developer.md](https://github.com/iovisor/bcc/blob/master/…/docs/tutorial_bcc_python_developer.md)
- […/docs/reference_guide.md](https://github.com/iovisor/bcc/blob/master/…/docs/reference_guide.md)
- […/CONTRIBUTING-SCRIPTS.md](https://github.com/iovisor/bcc/blob/master/…/CONTRIBUTING-SCRIPTS.md)

**ply:** [https://github.com/iovisor/ply](https://github.com/iovisor/ply)

**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)

**Netflix Tech Blog on Vector:**

**Linux Performance:** [http://www.brendangregg.com/linuxperf.html](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!