How to Mesure Linux Performance Wrong

… and right

Peter Zaitsev, CEO Percona

August 8th, 2019

Triangle Linux Users Group
Raleigh,NC

PERCONA

About Percona

Open Source Database Solutions Company

Support, Managed Services, Consulting, Training, Engineering

Focus on MySQL, MariaDB, MongoDB, PostgreSQL

Support Cloud DBaaS Variants on major clouds

Develop Database Software and Tools

Release Everything as 100% Free and Open Source

Widely Deployed Open Source Software

PERCONA
Server for MySQL

5,000,000+ downloads

PERCONA
Monitoring and Management

175,000+ downloads

PERCONA
XtraBackup

4,500,000+ downloads

PERCONA
Server for MongoDB

450,000+ downloads

PERCONA
Toolkit

2,000,000+ downloads

PERCONA
XtraDB Cluster

1,500,000+ downloads

What it has to do with Linux ?

95%+ of High Performance Open Source Databases Deployments are done on Linux

Personally has been running Linux since 1999

About You

Who are you ?

What is your interest in Linux Performance ?

About Presentation

Linux Performance Basics

Typical Mistakes and Right way to Look at the Problem

Cool new Stuff coming up

Percona Monitoring and Management

- 100% Free and Open Source
- Purpose Build for Open Source Database Monitoring
- Based on leading Open Source Technologies – Grafana, Prometheus
- Easy to Set up

PERCONA
Monitoring and Management

Linux Performance Basics

Linux Performance or Application Performance ?

It is Application Performance what is important in most cases

Bad Application will not Perform even on best tuned Linux Server

Linux Performance

Linux itself is not most typical cause of performance issues

Any Application can be impacted

But not every application will be impacted

When do you need to measure Performance ?

Troubleshooting

Capacity Planning

Cost and Efficiency Optimization

Change Management

Most Important Hardware Resources

CPU

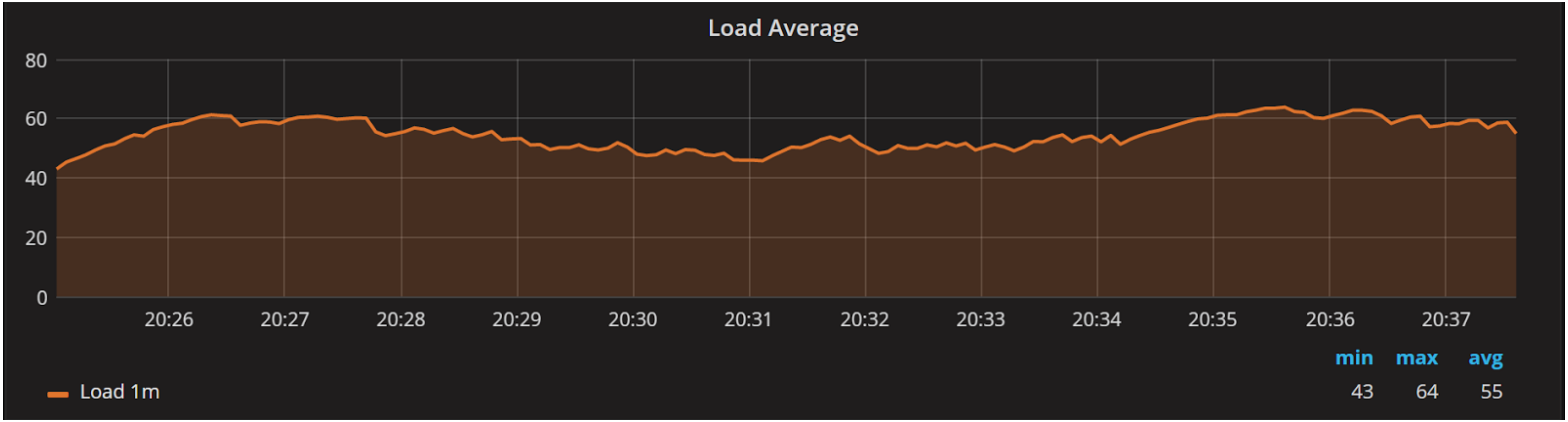
Memory

Disk

Network

Wrongs (and Rights) of Mesuring Linux Performance

#1 Focusing on LoadAvg



	min	max	avg
Load 1m	43	64	55

Problems with LoadAvg

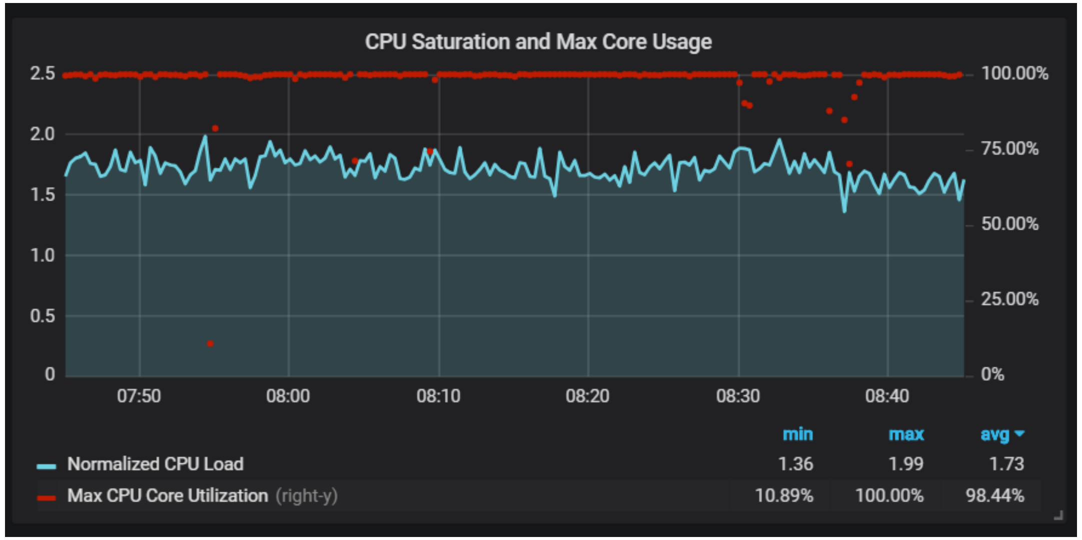
Mixes Apples and Oranges (CPU, Disk, Uninterruptable sleep)

Not Normalized

Exponential Moving Average

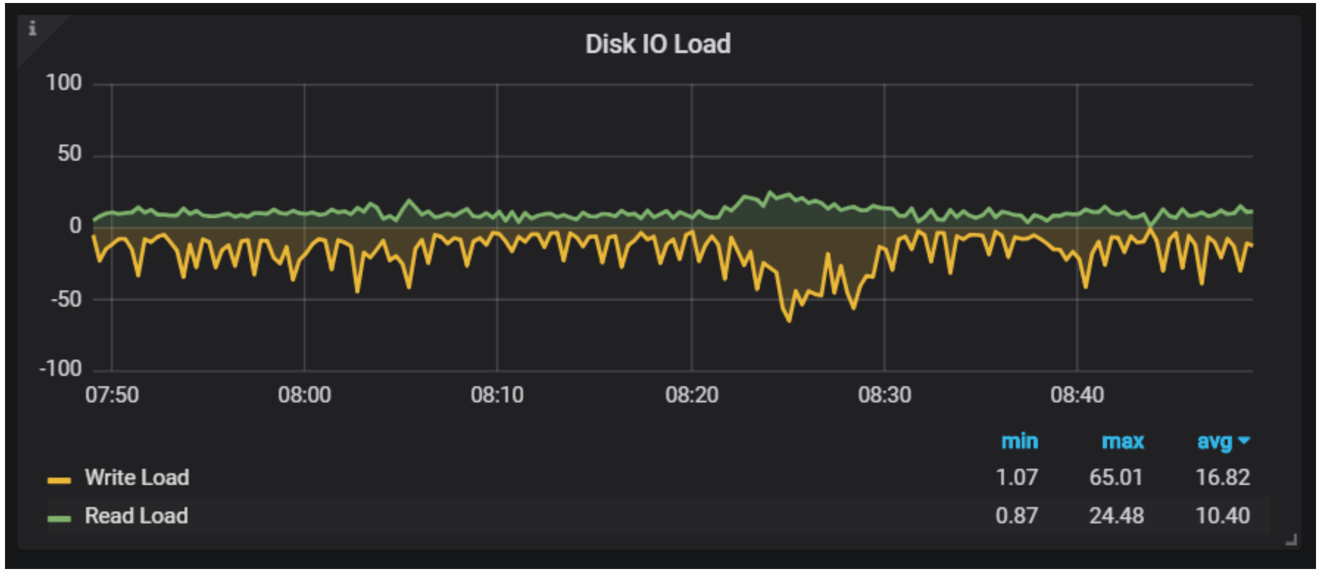
http://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html

Decomposing LoadAvg



CPU Saturation and Max Core Usage

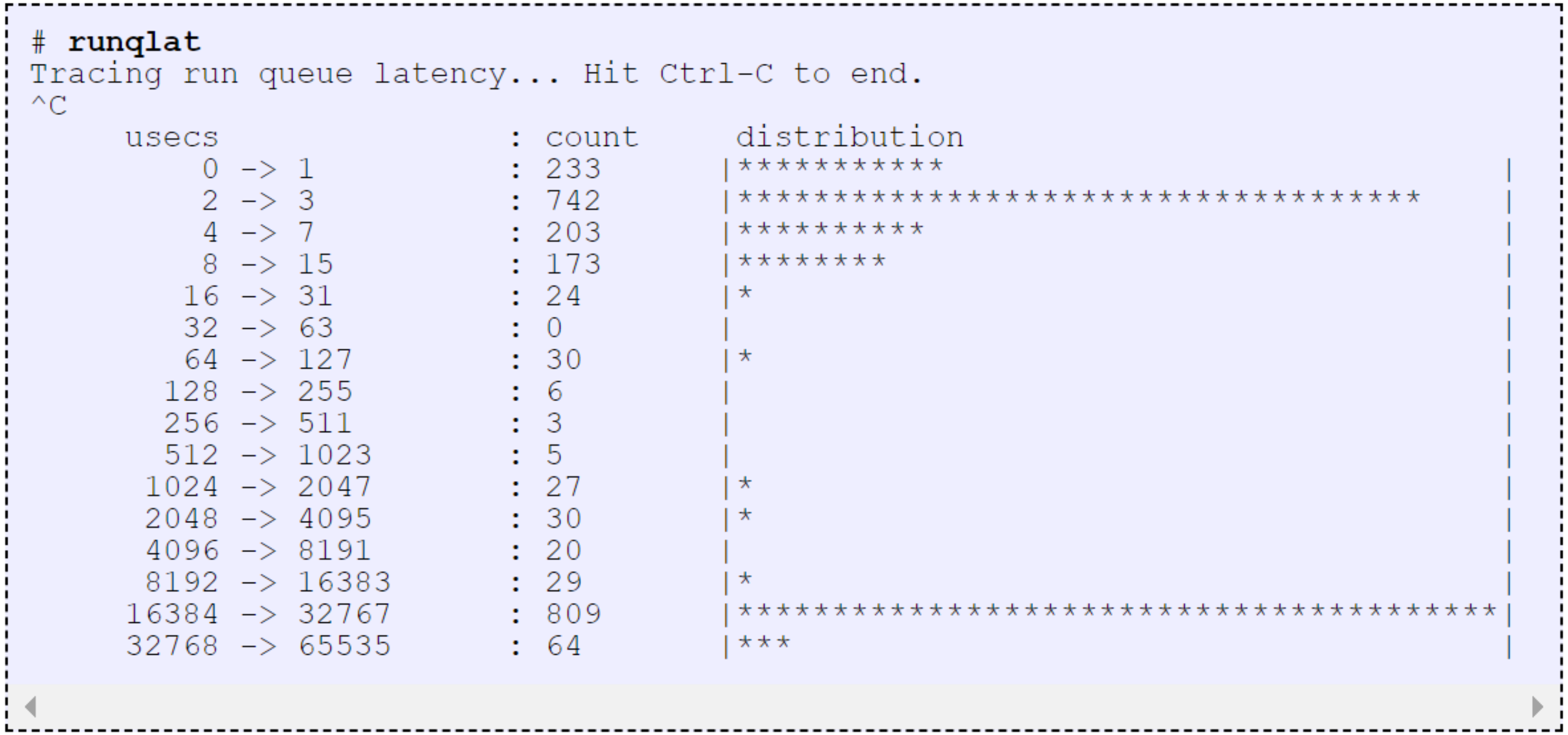
	min	max	avg
🟦 Normalized CPU Load	1.36	1.99	1.73
🟥 Max CPU Core Utilization (right-y)	10.89%	100.00%	98.44%



Disk IO Load

	min	max	avg
🟨 Write Load	1.07	65.01	16.82
🟩 Read Load	0.87	24.48	10.40

Run Queue Latency with BPF

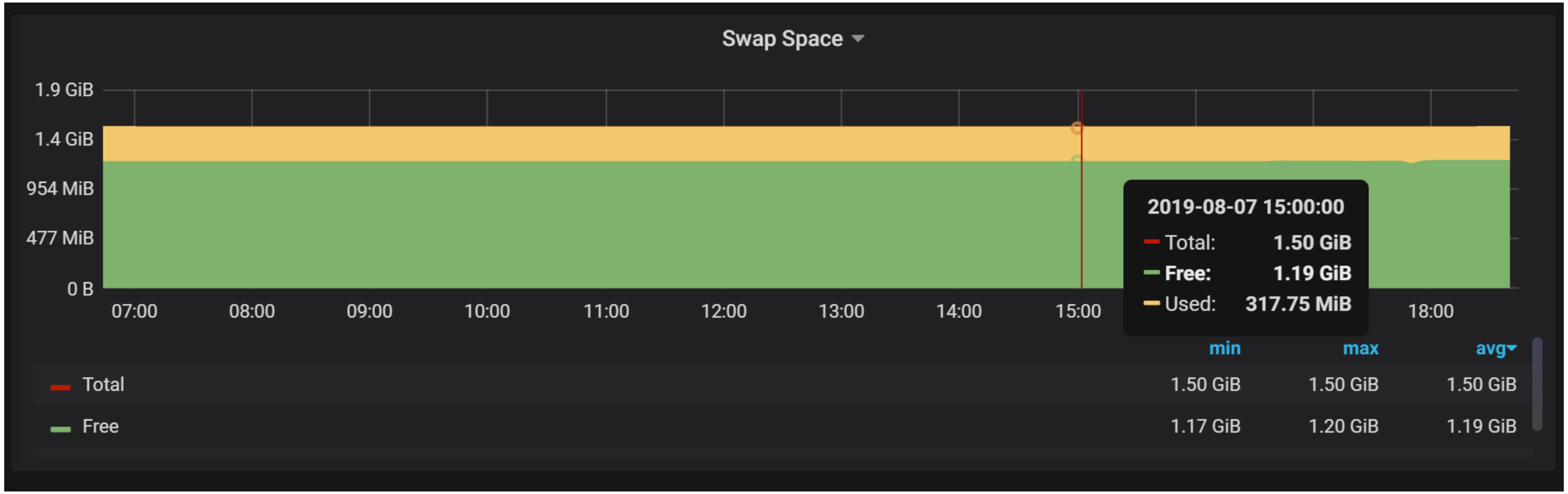


```
# runqlat
Tracing run queue latency... Hit Ctrl-C to end.
^C
     usecs               : count     distribution
         0 -> 1          : 233      |***********                             |
         2 -> 3          : 742      |************************************    |
         4 -> 7          : 203      |**********                              |
         8 -> 15         : 173      |********                                |
        16 -> 31         : 24       |*                                       |
        32 -> 63         : 0        |                                        |
        64 -> 127        : 30       |*                                       |
       128 -> 255        : 6        |                                        |
       256 -> 511        : 3        |                                        |
       512 -> 1023       : 5        |                                        |
      1024 -> 2047       : 27       |*                                       |
      2048 -> 4095       : 30       |*                                       |
      4096 -> 8191       : 20       |                                        |
      8192 -> 16383      : 29       |*                                       |
     16384 -> 32767      : 809      |****************************************|
     32768 -> 65535      : 64       |***                                     |
```

http://www.brendangregg.com/blog/2016-10-08/linux-bcc-runqlat.html

#2 Obcessing with Used Swap Space

- Used Swap Space is not reason to Panic
- There is some Never Used “Garbage” which is better in Swap Space

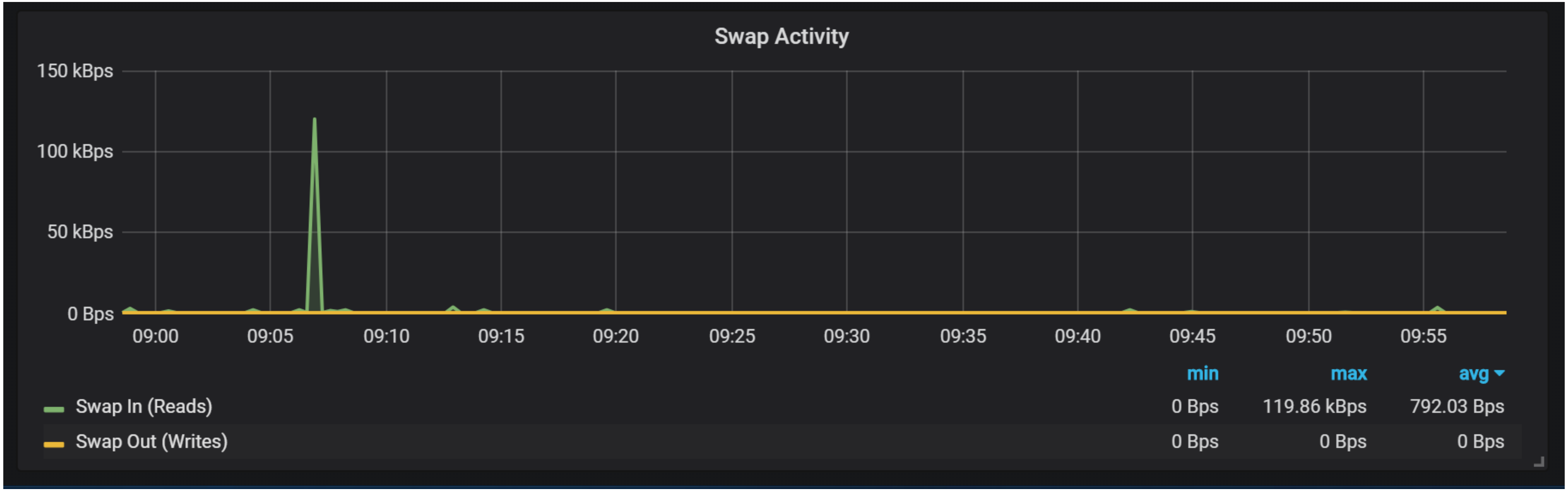


2019-08-07 15:00:00

- Total: 1.50 GiB
- **Free:** 1.19 GiB
- Used: 317.75 MiB

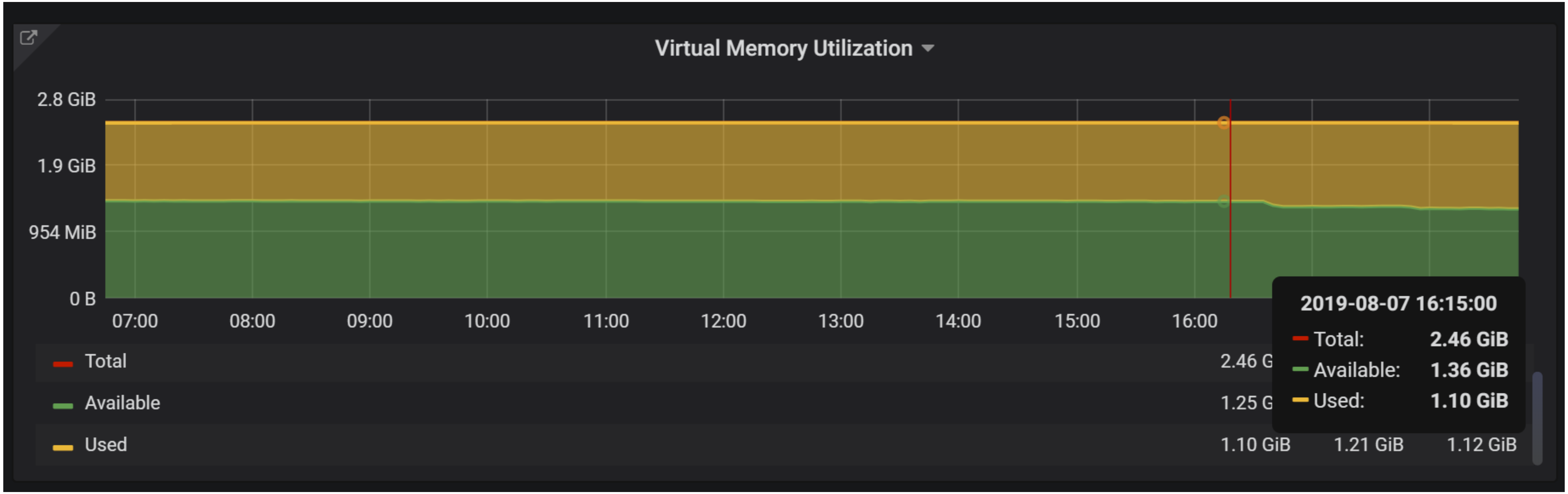
	min	max	avg
🟥 Total	1.50 GiB	1.50 GiB	1.50 GiB
🟩 Free	1.17 GiB	1.20 GiB	1.19 GiB

Better Way: Look at the Swap IO



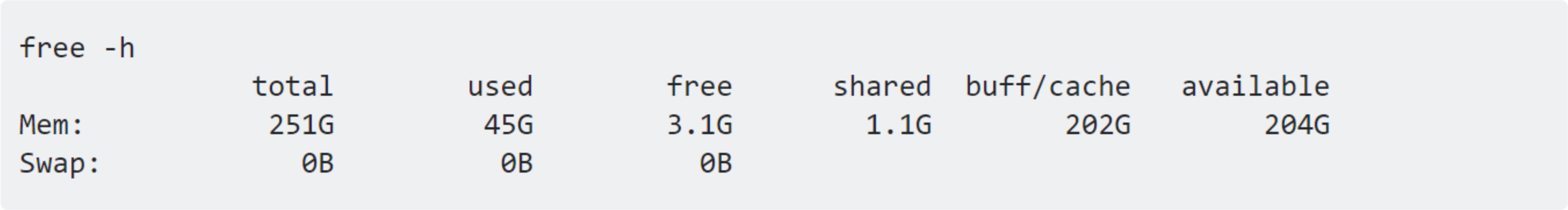
	min	max	avg ▾
🟩 Swap In (Reads)	0 Bps	119.86 kBps	792.03 Bps
🟨 Swap Out (Writes)	0 Bps	0 Bps	0 Bps

.. And Available Virtual Memory



#3 Being Concerned about “Free” Memory

- Linux will use memory for caching, look for “Available” instead



```
free -h
              total        used        free      shared  buff/cache   available
Mem:           251G         45G        3.1G        1.1G        202G        204G
Swap:            0B          0B          0B
```

#4 Confusing Throughput with Latency

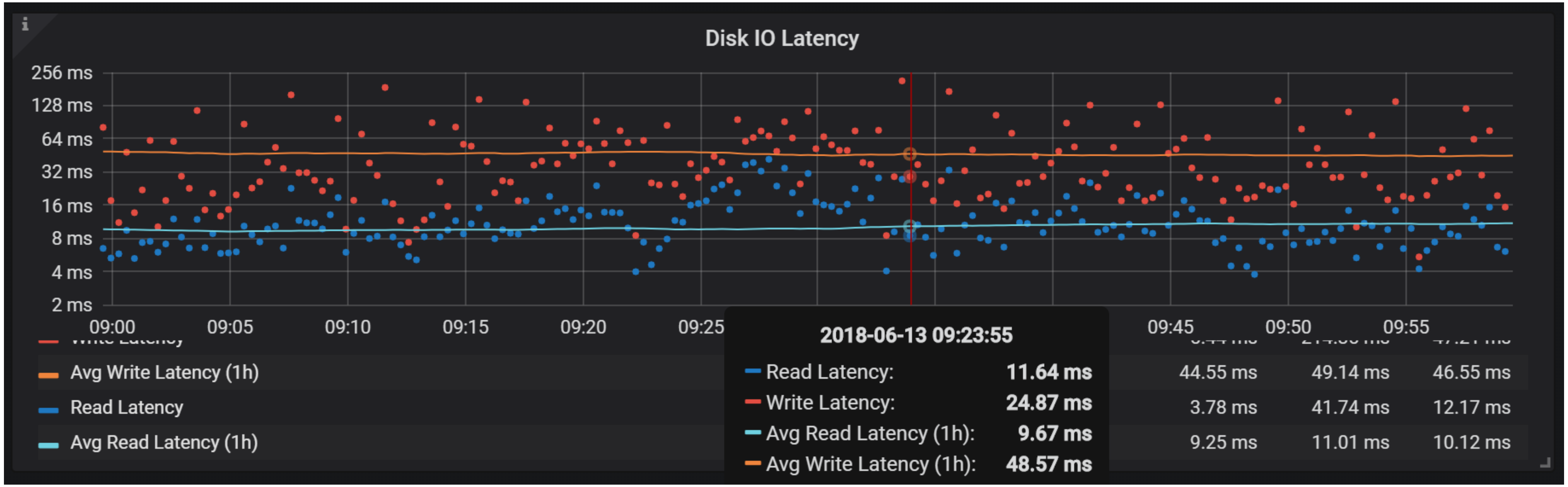
Excited your IO Subsystem can do 10K IOPS

Do not forget to ask about Latency

SAN, Cloud Storage often has very good throughput but poor latency

#5 Mixing Read and Write Latencies Together

- Modern Storage can have very different paths for reads and writes



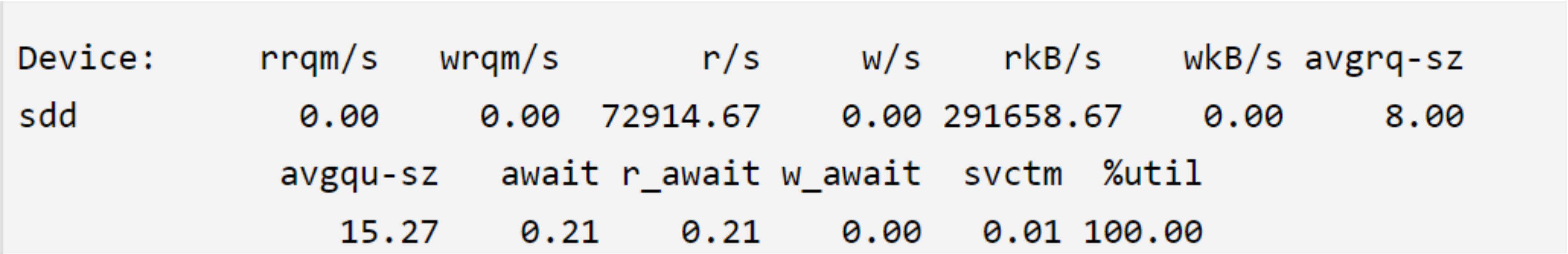
Avg Write Latency (1h)	44.55 ms	49.14 ms	46.55 ms
Read Latency	3.78 ms	41.74 ms	12.17 ms
Avg Read Latency (1h)	9.25 ms	11.01 ms	10.12 ms

2018-06-13 09:23:55

Read Latency:	**11.64 ms**
Write Latency:	**24.87 ms**
Avg Read Latency (1h):	**9.67 ms**
Avg Write Latency (1h):	**48.57 ms**

#6 Using iostat “utilization” metric

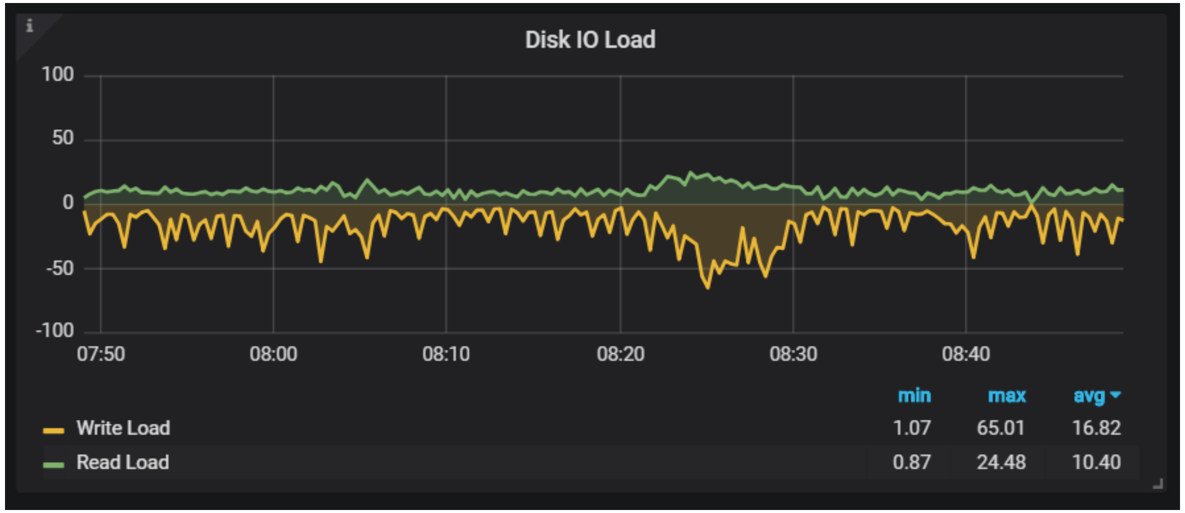
- Low Utilization means drive is not heavily used
- High Utilization … means Little



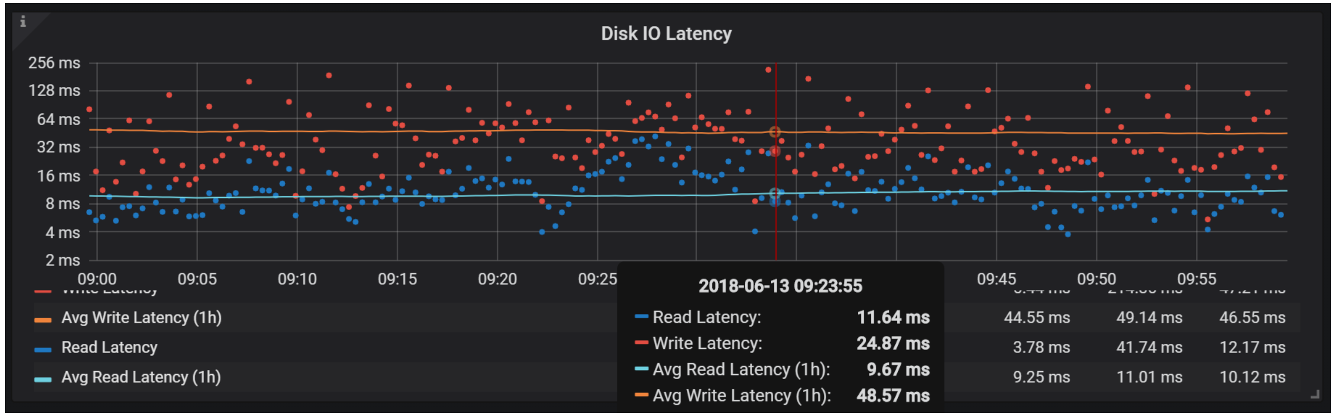
```
Device:         rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s avgrq-sz
sdd               0.00     0.00 72914.67    0.00 291658.67     0.00     8.00
                avgqu-sz   await r_await w_await  svctm  %util
                   15.27    0.21    0.21    0.00   0.01 100.00
```

https://brooker.co.za/blog/2014/07/04/iostat-pct.html

Better Way ?



	min	max	avg ▾
Write Load	1.07	65.01	16.82
Read Load	0.87	24.48	10.40



Write Latency	[illegible]	[illegible]	[illegible]
Avg Write Latency (1h)	44.55 ms	49.14 ms	46.55 ms
Read Latency	3.78 ms	41.74 ms	12.17 ms
Avg Read Latency (1h)	9.25 ms	11.01 ms	10.12 ms

2018-06-13 09:23:55

Read Latency:	11.64 ms
Write Latency:	24.87 ms
Avg Read Latency (1h):	9.67 ms
Avg Write Latency (1h):	48.57 ms

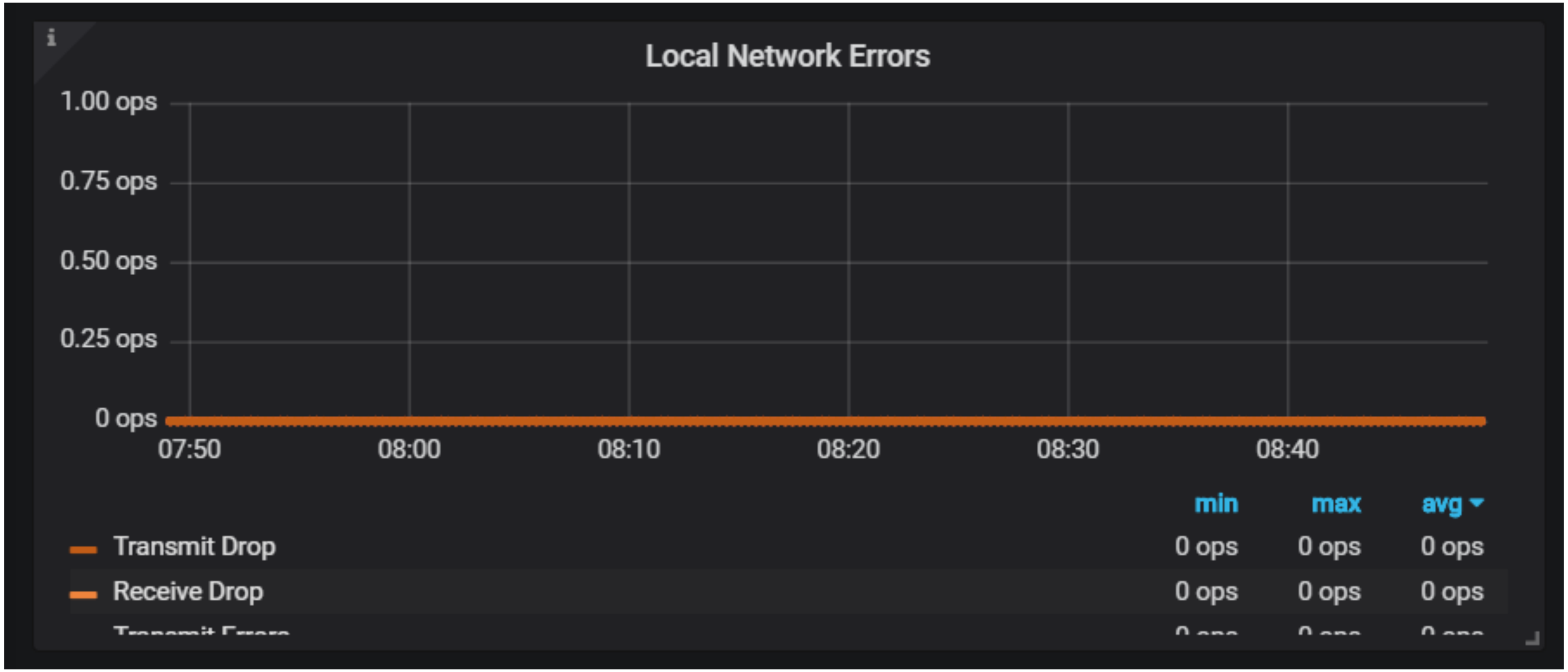
PERCONA

#7 Thinking Network is about Local Bandwidth

You have 10GB connection… but what about Oversubscription on Switches ?

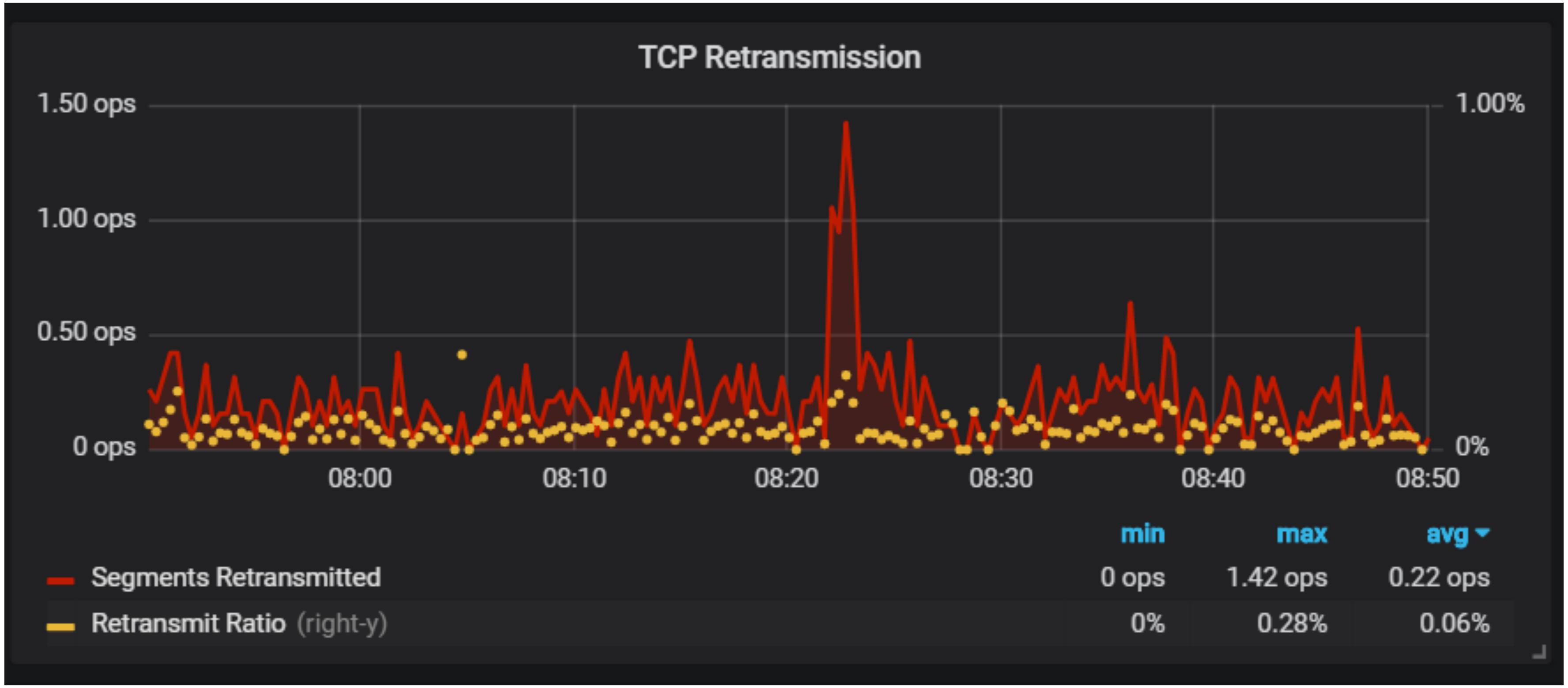
Consider Latency which comes from Distance and Routing Devices

#8 Forgetting to check Local Network Status



	min	max	avg
Transmit Drop	0 ops	0 ops	0 ops
Receive Drop	0 ops	0 ops	0 ops
Transmit Errors	0 ops	0 ops	0 ops

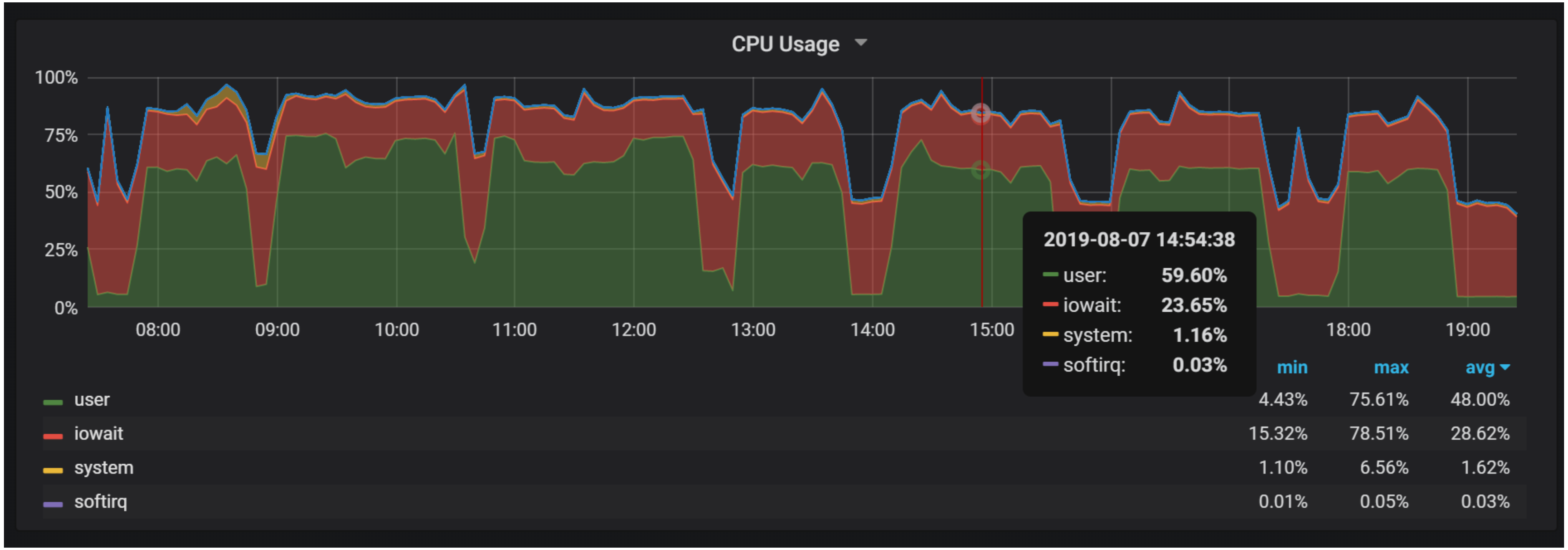
#9 Misunderstanding Retransmits



	min	max	avg
Segments Retransmitted	0 ops	1.42 ops	0.22 ops
Retransmit Ratio (right-y)	0%	0.28%	0.06%

#10 Including IOWait in CPU Utilization

- “Everything which is not Idle is CPU Used”
- IOWait is type of Idle, when it is idle due to some of disk waits

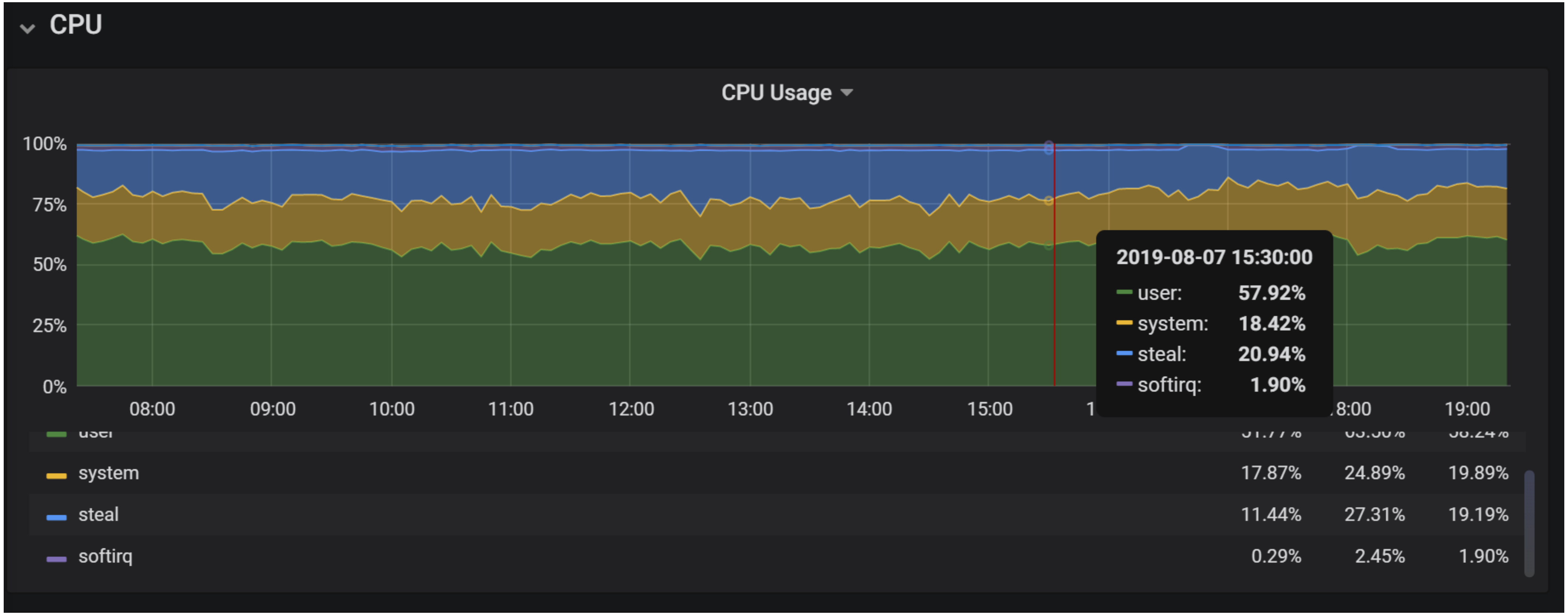


2019-08-07 14:54:38
- user: 59.60%
- iowait: 23.65%
- system: 1.16%
- softirq: 0.03%

	min	max	avg
user	4.43%	75.61%	48.00%
iowait	15.32%	78.51%	28.62%
system	1.10%	6.56%	1.62%
softirq	0.01%	0.05%	0.03%

#11 Ignoring “Steal”

- **Very Important with Virtualization and Cloud**



2019-08-07 15:30:00

user:	57.92%
system:	18.42%
steal:	20.94%
softirq:	1.90%

user	[illegible]	[illegible]	[illegible]
system	17.87%	24.89%	19.89%
steal	11.44%	27.31%	19.19%
softirq	0.29%	2.45%	1.90%

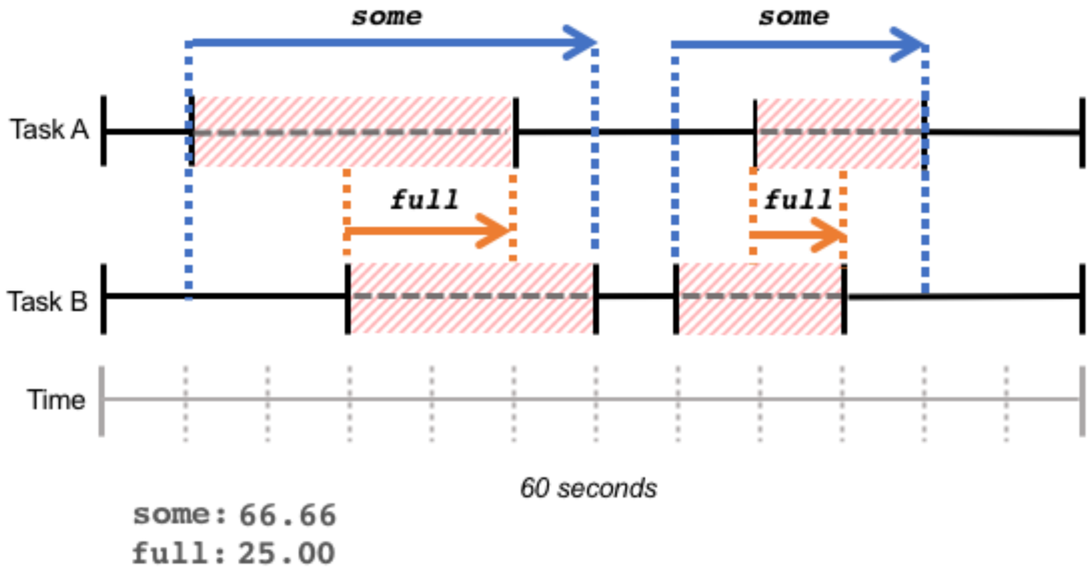
What would you add ?

What mistakes have you seen ?

Cool Stuff Coming up

/proc/pressure

- Available in Linux Kernel 4.20+
- Measure “Pressure” on CPU, Memory, Disk as the time process waited on those resources



```
some: 66.66
full: 25.00
```

https://facebookmicrosites.github.io/psi/docs/overview

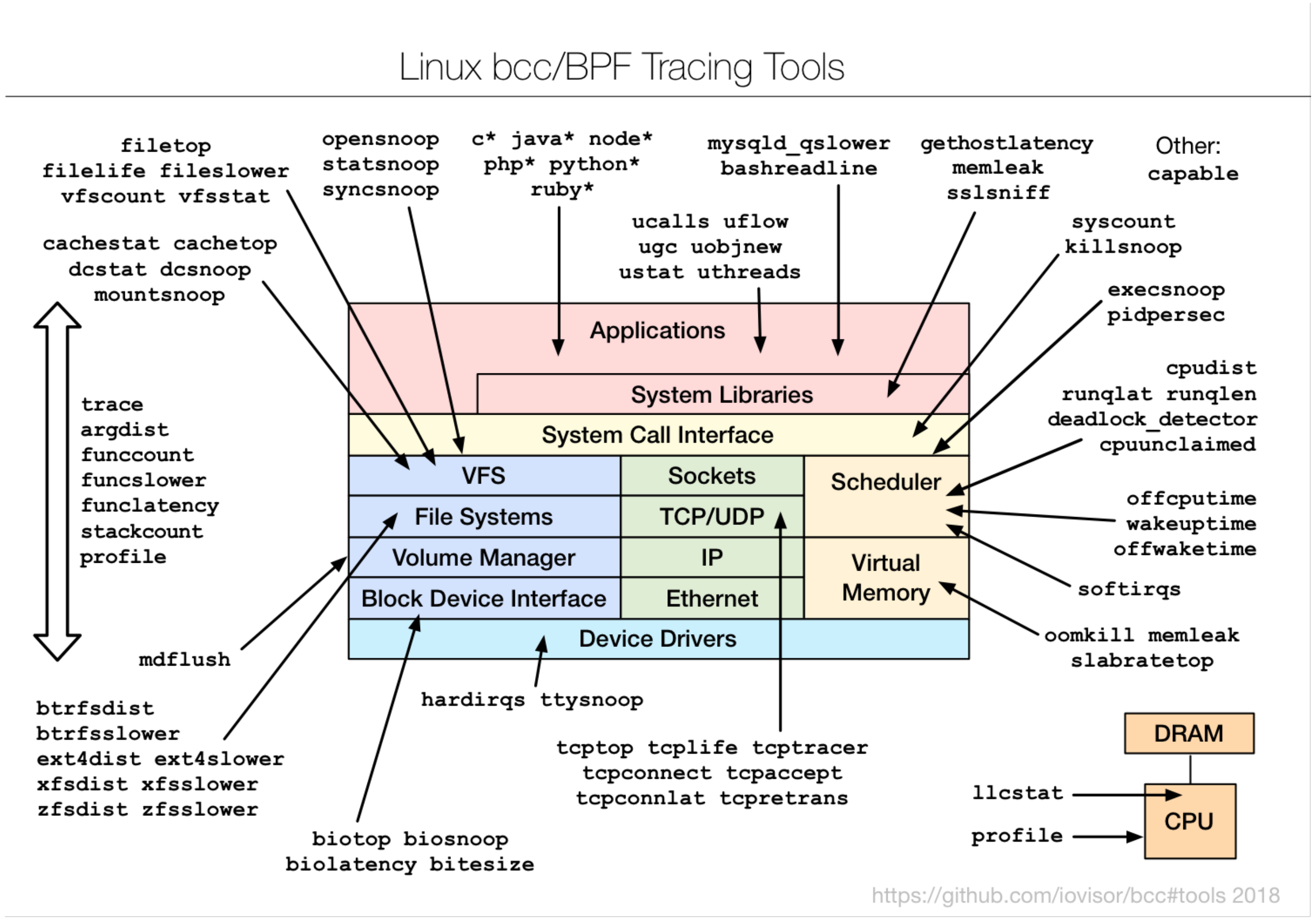
eBPF in Linux

Not new, Have been in mainline since 2014

Actively improved

Decent availability in Linux Distributions

eBPF in Linux Summary



Linux bcc/BPF Tracing Tools

filetop
filelife fileslower
vfscount vfsstat

opensnoop
statsnoop
syncsnoop

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

mysqld_qslower
bashreadline

gethostlatency
memleak
sslsniff

Other:
capable

cachestat cachetop
dcstat dcsnoop
mountsnoop

ucalls uflow
ugc uobjnew
ustat uthreads

syscount
killsnoop

execsnoop
pidpersec

trace
argdist
funccount
funcslower
funclatency
stackcount
profile

cpudist
runqlat runqlen
deadlock_detector
cpuunclaimed

Applications

System Libraries

System Call Interface

VFS

Sockets

Scheduler

File Systems

TCP/UDP

Volume Manager

IP

Virtual Memory

Block Device Interface

Ethernet

Device Drivers

offcputime
wakeuptime
offwaketime

softirqs

oomkill memleak
slabratetop

mdflush

btrfsdist
btrfsslower
ext4dist ext4slower
xfsdist xfsslower
zfsdist zfsslower

hardirqs ttysnoop

tcptop tcplife tcptracer
tcpconnect tcpaccept
tcpconnlat tcpretrans

biotop biosnoop
biolatency bitesize

DRAM

llcstat

CPU

profile

https://github.com/iovisor/bcc#tools 2018

eBPF Superpowers

Instead of Hardcoded counters placed through the Kernel

We can connect to any tracepoint

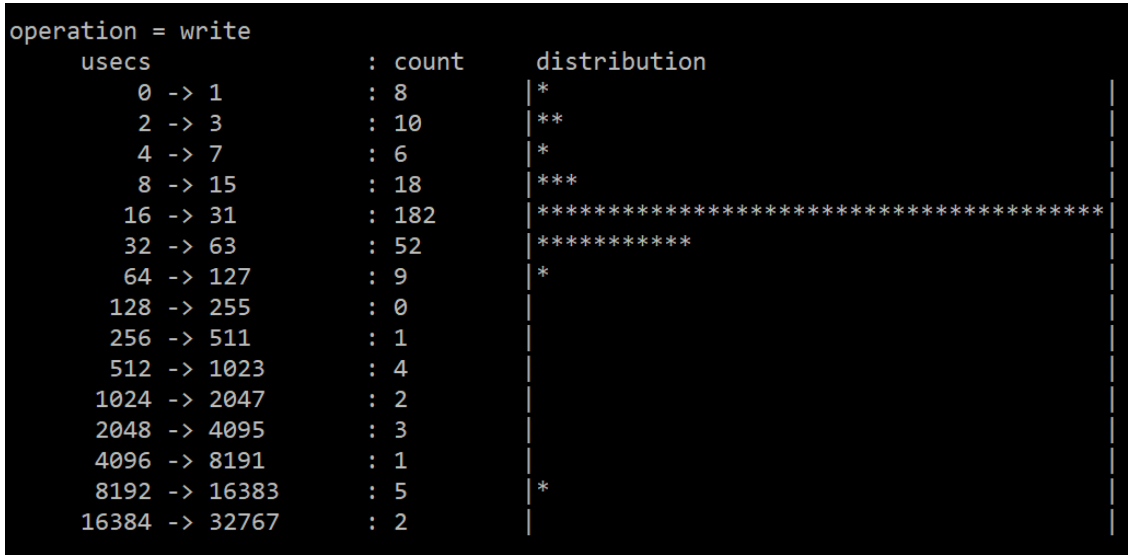
And process information in many different ways (ie histogram rather than counter)

With Great Power Comes Great Responsibility

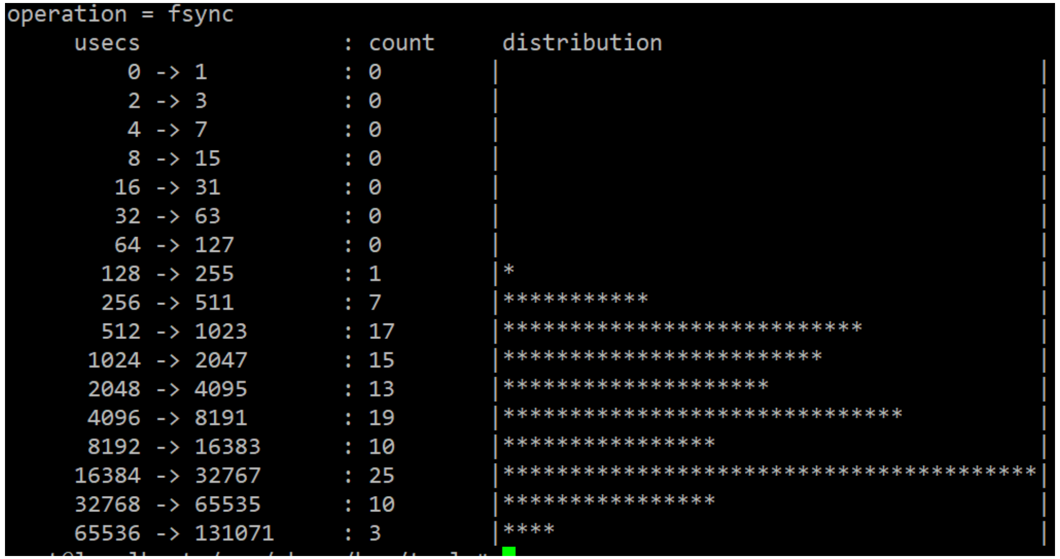
By connecting complicated eBPF Programs to frequently triggered tracepoints you can slow down your system dramatically

Kernel checks eBPF Programs to save you from many mistakes

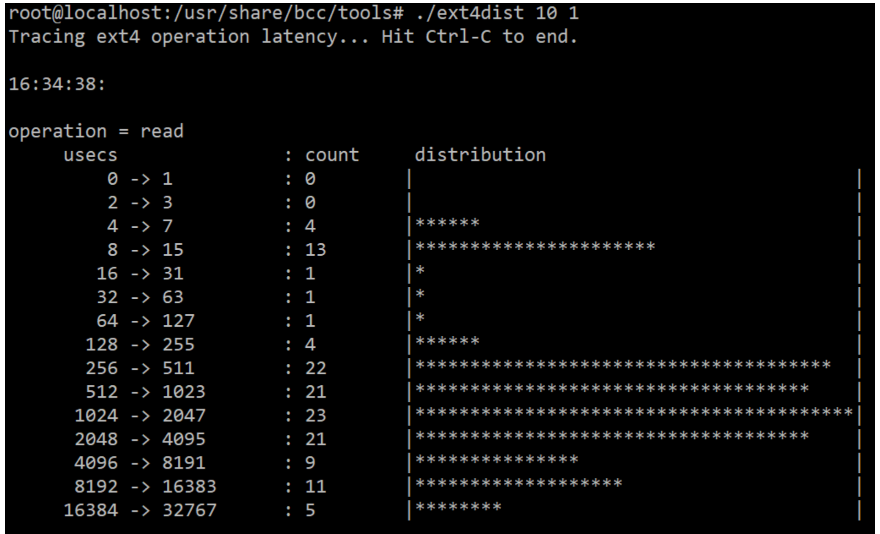
Ext4dist: Filesystem Latency per Operation



```
operation = write
     usecs               : count     distribution
         0 -> 1          : 8        |*                                       |
         2 -> 3          : 10       |**                                      |
         4 -> 7          : 6        |*                                       |
         8 -> 15         : 18       |***                                     |
        16 -> 31         : 182      |****************************************|
        32 -> 63         : 52       |***********                             |
        64 -> 127        : 9        |*                                       |
       128 -> 255        : 0        |                                        |
       256 -> 511        : 1        |                                        |
       512 -> 1023       : 4        |                                        |
      1024 -> 2047       : 2        |                                        |
      2048 -> 4095       : 3        |                                        |
      4096 -> 8191       : 1        |                                        |
      8192 -> 16383      : 5        |*                                       |
     16384 -> 32767      : 2        |                                        |
```

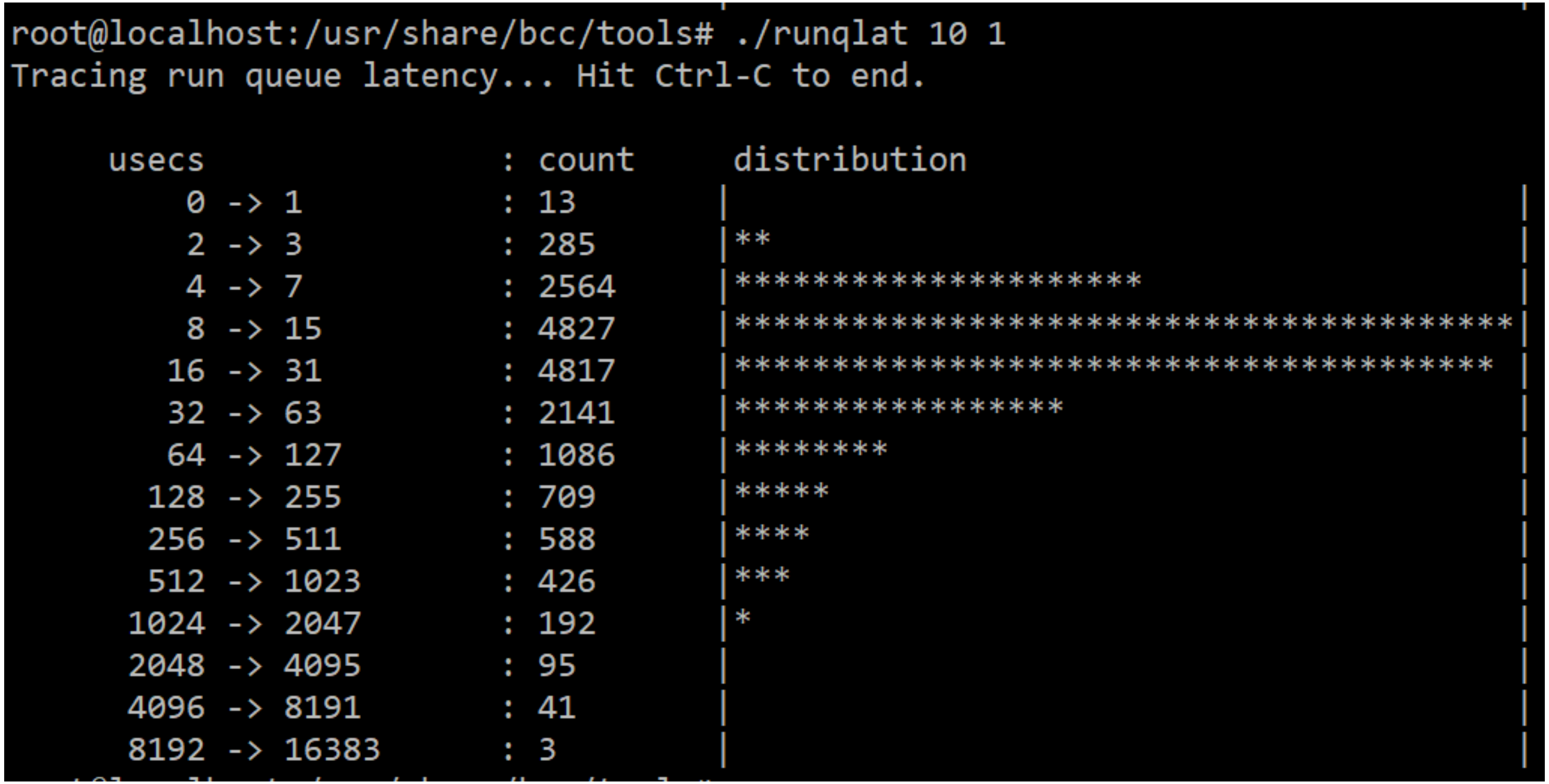


```
operation = fsync
     usecs               : count     distribution
         0 -> 1          : 0        |                                        |
         2 -> 3          : 0        |                                        |
         4 -> 7          : 0        |                                        |
         8 -> 15         : 0        |                                        |
        16 -> 31         : 0        |                                        |
        32 -> 63         : 0        |                                        |
        64 -> 127        : 0        |                                        |
       128 -> 255        : 1        |*                                       |
       256 -> 511        : 7        |***********                             |
       512 -> 1023       : 17       |***************************             |
      1024 -> 2047       : 15       |************************                |
      2048 -> 4095       : 13       |********************                    |
      4096 -> 8191       : 19       |******************************          |
      8192 -> 16383      : 10       |****************                        |
     16384 -> 32767      : 25       |****************************************|
     32768 -> 65535      : 10       |****************                        |
     65536 -> 131071     : 3        |****                                    |
```



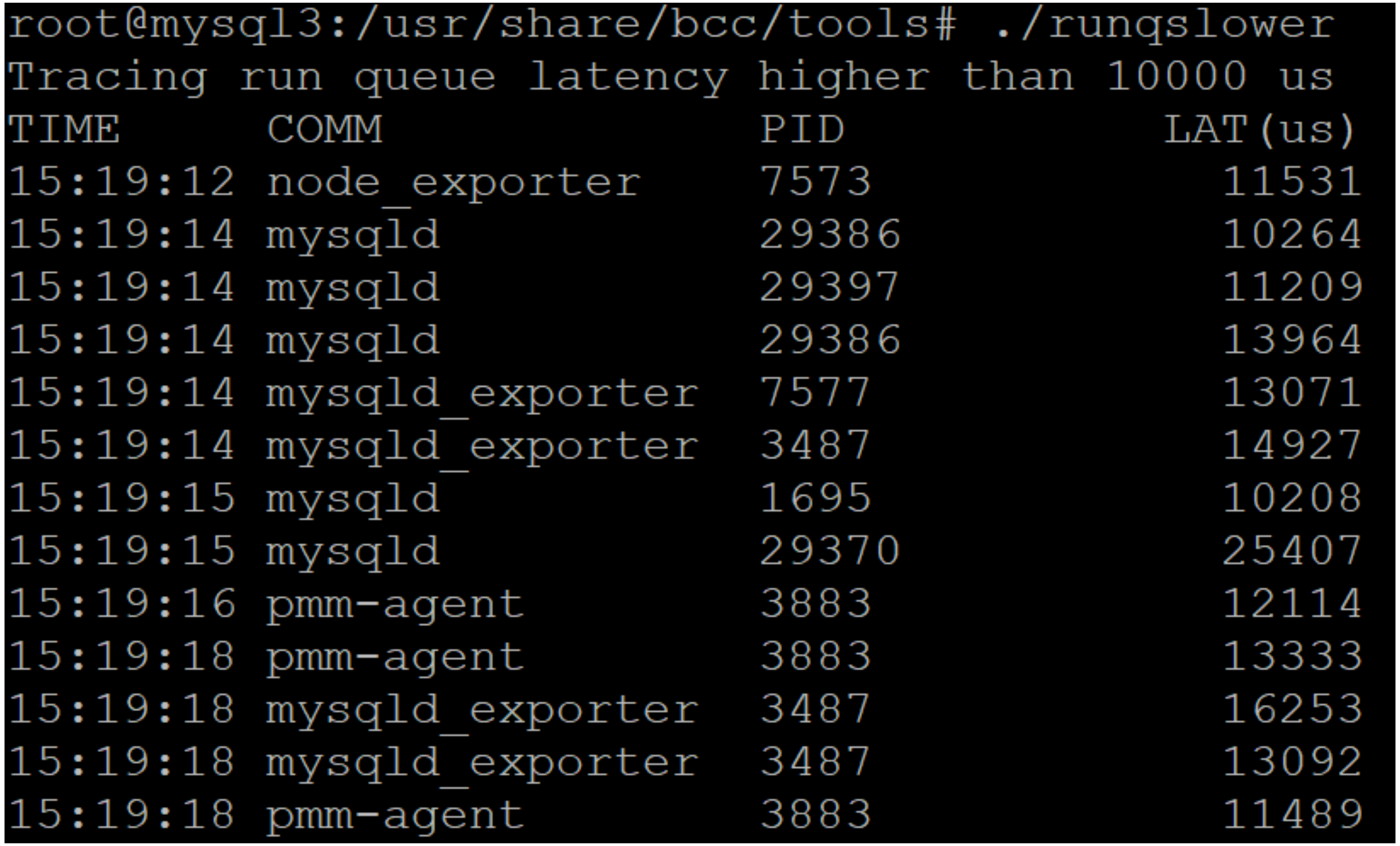
```
root@localhost:/usr/share/bcc/tools# ./ext4dist 10 1
Tracing ext4 operation latency... Hit Ctrl-C to end.

16:34:38:

operation = read
     usecs               : count     distribution
         0 -> 1          : 0        |                                        |
         2 -> 3          : 0        |                                        |
         4 -> 7          : 4        |******                                  |
         8 -> 15         : 13       |**********************                  |
        16 -> 31         : 1        |*                                       |
        32 -> 63         : 1        |*                                       |
        64 -> 127        : 1        |*                                       |
       128 -> 255        : 4        |******                                  |
       256 -> 511        : 22       |**************************************  |
       512 -> 1023       : 21       |************************************    |
      1024 -> 2047       : 23       |****************************************|
      2048 -> 4095       : 21       |************************************    |
      4096 -> 8191       : 9        |***************                         |
      8192 -> 16383      : 11       |*******************                     |
     16384 -> 32767      : 5        |********                                |
```

Runqlat: CPU RunQueue Latency



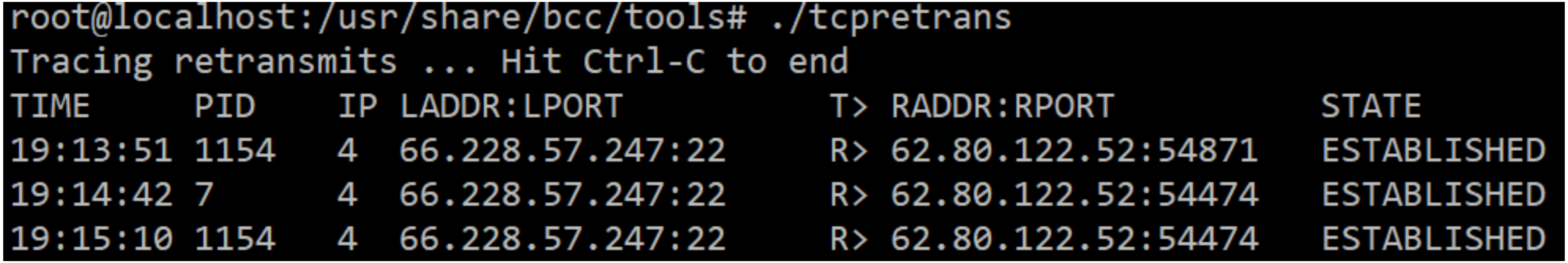
```
root@localhost:/usr/share/bcc/tools# ./runqlat 10 1
Tracing run queue latency... Hit Ctrl-C to end.

     usecs               : count     distribution
         0 -> 1          : 13       |                                        |
         2 -> 3          : 285      |**                                      |
         4 -> 7          : 2564     |*********************                   |
         8 -> 15         : 4827     |****************************************|
        16 -> 31         : 4817     |*************************************** |
        32 -> 63         : 2141     |*****************                       |
        64 -> 127        : 1086     |********                                |
       128 -> 255        : 709      |*****                                   |
       256 -> 511        : 588      |****                                    |
       512 -> 1023       : 426      |***                                     |
      1024 -> 2047       : 192      |*                                       |
      2048 -> 4095       : 95       |                                        |
      4096 -> 8191       : 41       |                                        |
      8192 -> 16383      : 3        |                                        |
```

Run queue Outliers



```
root@mysql3:/usr/share/bcc/tools# ./runqslower
Tracing run queue latency higher than 10000 us
TIME     COMM                 PID            LAT(us)
15:19:12 node_exporter        7573             11531
15:19:14 mysqld               29386            10264
15:19:14 mysqld               29397            11209
15:19:14 mysqld               29386            13964
15:19:14 mysqld_exporter      7577             13071
15:19:14 mysqld_exporter      3487             14927
15:19:15 mysqld               1695             10208
15:19:15 mysqld               29370            25407
15:19:16 pmm-agent            3883             12114
15:19:18 pmm-agent            3883             13333
15:19:18 mysqld_exporter      3487             16253
15:19:18 mysqld_exporter      3487             13092
15:19:18 pmm-agent            3883             11489
```

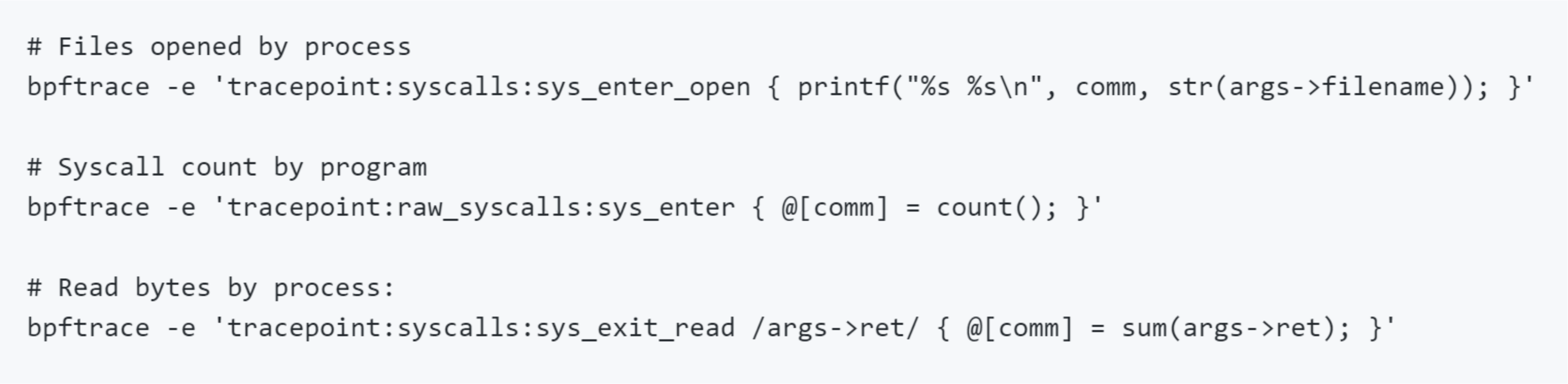
Tcpretrans: TCP Retransmits Details



```
root@localhost:/usr/share/bcc/tools# ./tcpretrans
Tracing retransmits ... Hit Ctrl-C to end
TIME     PID    IP LADDR:LPORT          T> RADDR:RPORT          STATE
19:13:51 1154   4  66.228.57.247:22     R> 62.80.122.52:54871   ESTABLISHED
19:14:42 7      4  66.228.57.247:22     R> 62.80.122.52:54474   ESTABLISHED
19:15:10 1154   4  66.228.57.247:22     R> 62.80.122.52:54474   ESTABLISHED
```

BPFTrace

- Dtrace “Frontend” Alternative for Linux
- Simple Programming Language
- Powerful One-liners



```
# Files opened by process
bpftrace -e 'tracepoint:syscalls:sys_enter_open { printf("%s %s\n", comm, str(args->filename)); }'

# Syscall count by program
bpftrace -e 'tracepoint:raw_syscalls:sys_enter { @[comm] = count(); }'

# Read bytes by process:
bpftrace -e 'tracepoint:syscalls:sys_exit_read /args->ret/ { @[comm] = sum(args->ret); }'
```

https://github.com/iovisor/bpftrace

Check out eBPF Bible

http://www.brendangregg.com/ebpf.html

OPEN SOURCE DATABASE CONFERENCE

30 Sept - 2 Oct 2019

PERCONA LIVE EUROPE

AMSTERDAM

Become a part of the vibrant open source community. **Join Us!**

Thank You!

@PeterZaitsev

https://www.linkedin.com/in/peterzaitsev/