



linux and glibc: The 4.5TiB malloc API trace

Presented at Linux Plumbers Conference 2016

DJ Delorie, Carlos O'Donnell, Florian Weimer
2016-11-03

Disclaimer

Really really really don't run this in production

- Presenting experimental results
- Presenting about experimental code in glibc “dj/malloc” branch
- Don't use in production!

Available right now!

Right now.

Trace, conversion, and simulation on “dj/malloc”:

```
git clone git://sourceware.org/git/glibc.git
```

Data Analysis Tooling:

```
https://pagure.io/glibc-malloc-trace-utils
```

Overview

- Whole-system trace and benchmarking
- API tracing
- Trace to workload conversion
- Workload simulation

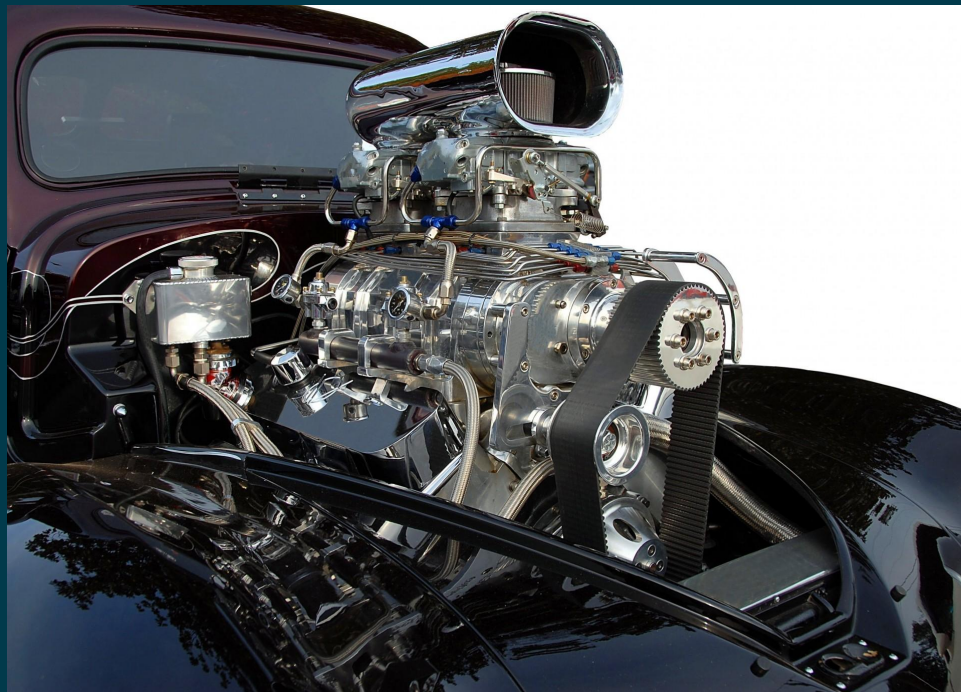
Whole-system benchmarking

Background.

- What problem are we trying to solve?
 - **First:** patch review.
 - **Second:** performance tracking release to release.
 - **Third:** ... helping developers find problems?
- A glibc whole-system benchmark is a dataset that characterizes a user workload and is used to test the behaviour of a change, but across a wider set of APIs i.e. a whole system.

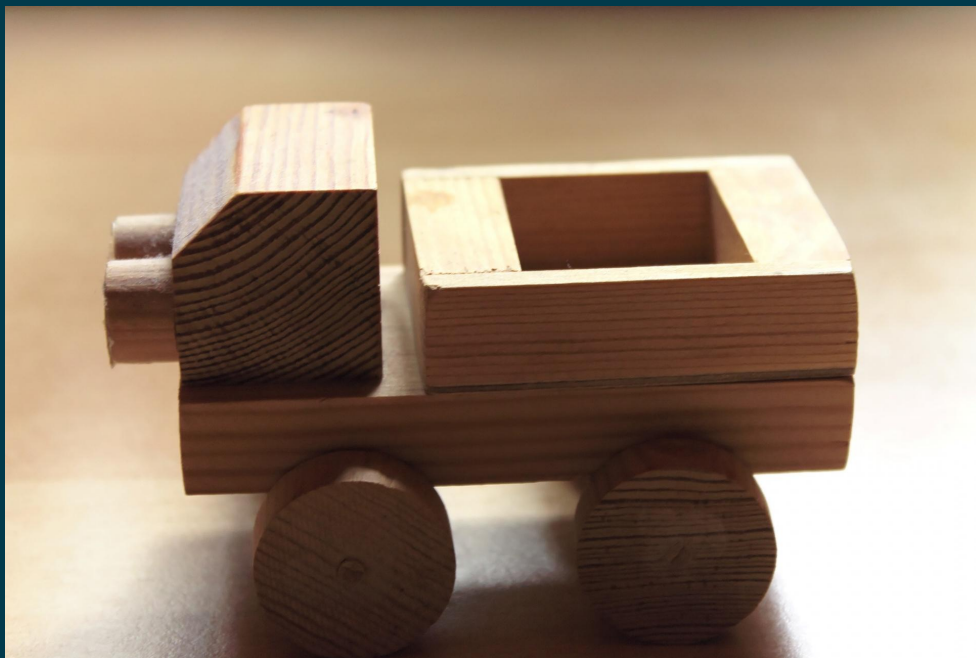
Whole-system benchmarking

The entire system is complicated.



Whole-system benchmarking

The malloc API is a smaller more tractable problem.



API tracing:

Original Goals

Support development of thread-local cache in glibc malloc:

- As low overhead as possible
- Be able to prove that a code path is taken (coverage, debugging)
- Determine which code paths are “hot” vs “cold” (performance)
- Reproduce difficult-to-automate scenarios (coverage)
- Represent many interests when profiling (performance)

API tracing:

NIH?

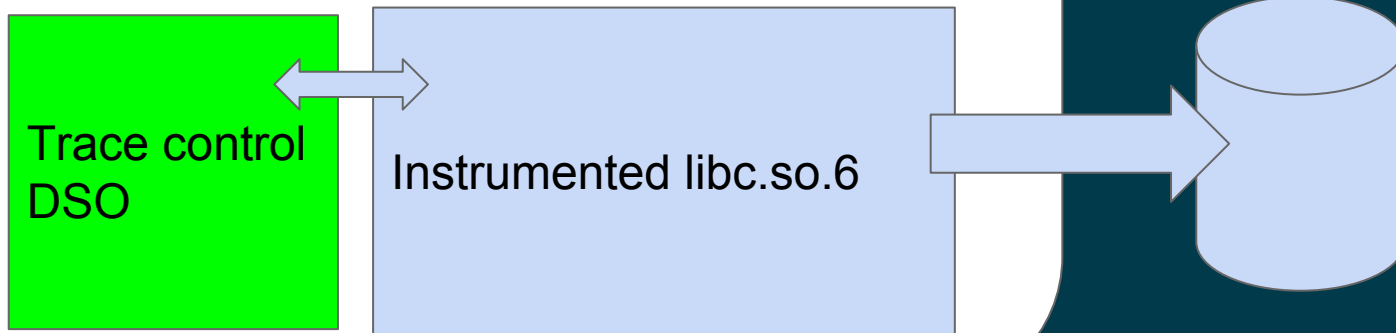
What else could we have used?

- systemtap (cost)
- dyninst (prototype difficult, future direction though)
- LTTng (cost)
- LTTng-ust (theoretical event loss, future direction)
- ftrace (cost, future direction)
- kprobes/uprobes (cost)

Application:

Calls malloc, free, calloc...

```
mtrace_init  
mtrace_start  
mtrace_stop  
mtrace_pause  
mtrace_unpause  
mtrace_sync  
mtrace_reset
```



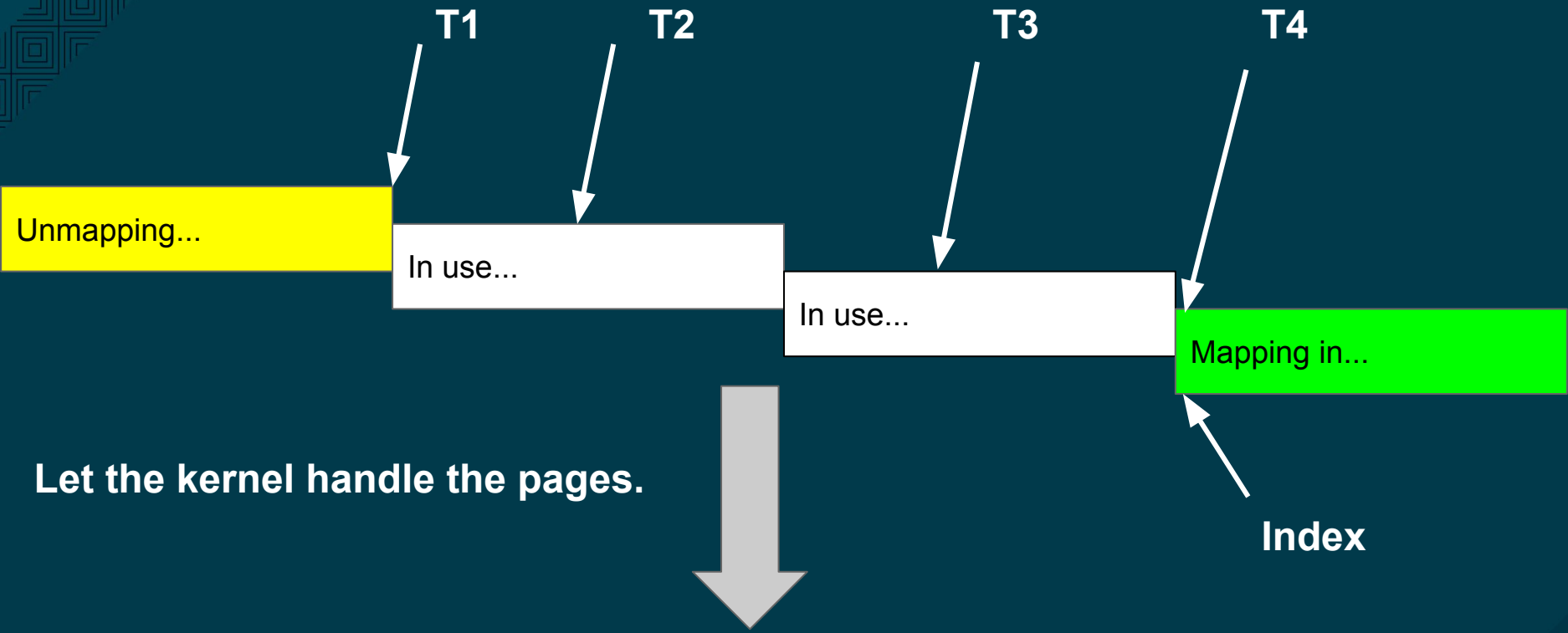
Tracing:

What do we trace?

We capture one trace record per API call (malloc, free, etc)

- Thread ID
- Call type (malloc, free, etc)
- Code paths (hot paths vs cold, hints about syscalls etc)
- Passed and returned pointers and sizes
- Internal information (available size, for overhead calculations)

The trace is a binary record streamed to a file while the applications runs.

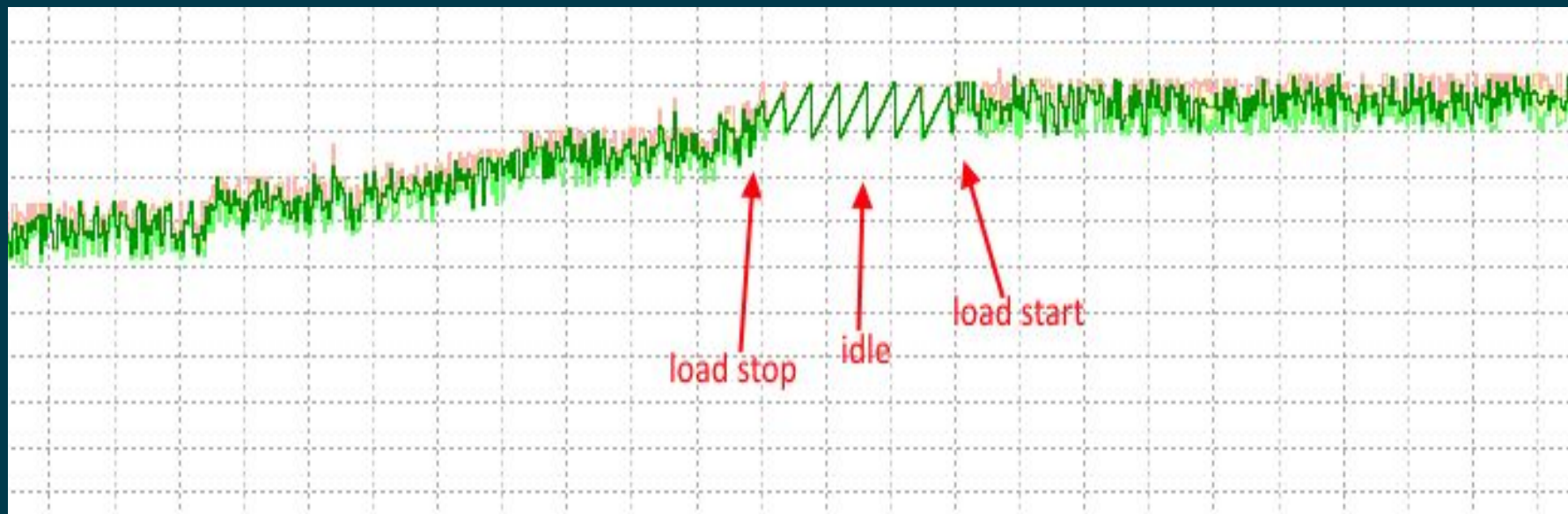


Let the kernel handle the pages.

On disk binary trace...

Tracing:

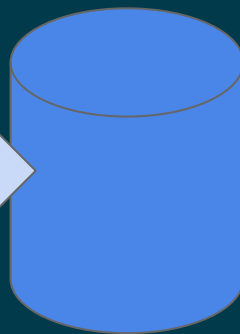
In process RSS changes



Raw Trace:



Workload File:



- Threads
- Sync
- Calls
- Args

T1 (Thread)

```
ptr_1 = malloc (...);
```

(Sync)

```
ptr_2 = calloc (...); (Args)
```

```
free (...); (Calls)
```

...

T2 (Thread)

```
free (ptr_1);
```

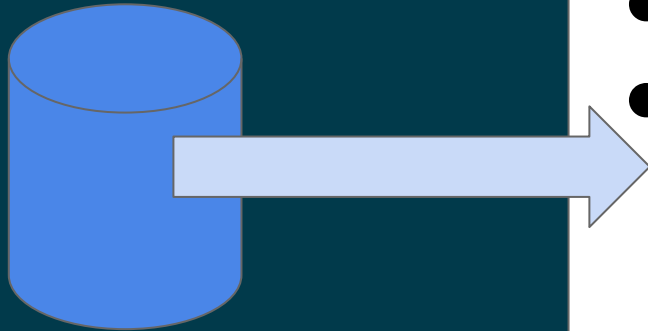
```
ptr_3 = malloc (...); (idx3)
```

...

```
free (ptr_3); (idx3)
```

Workload File:

Simulation:



- Multi-threaded
- Synchronizes
- Simulate API calls

Library under test:
glibc, jemalloc,
tcmalloc...

Simulation results

What and how...

683,978,658,689,650 cycles

302,522,994,686 usec wall time

416,319,364,837 usec across 50 threads

242,515,968 bytes Max RSS

(67,071,483,904 -> 67,313,999,872)

...

153,649 Kb Max Ideal RSS

Avg malloc time: 400 in 12,272,385,738 calls

Avg calloc time: 86,012 in 1,041,925 calls

Avg realloc time: 2,022 in 4,489 calls

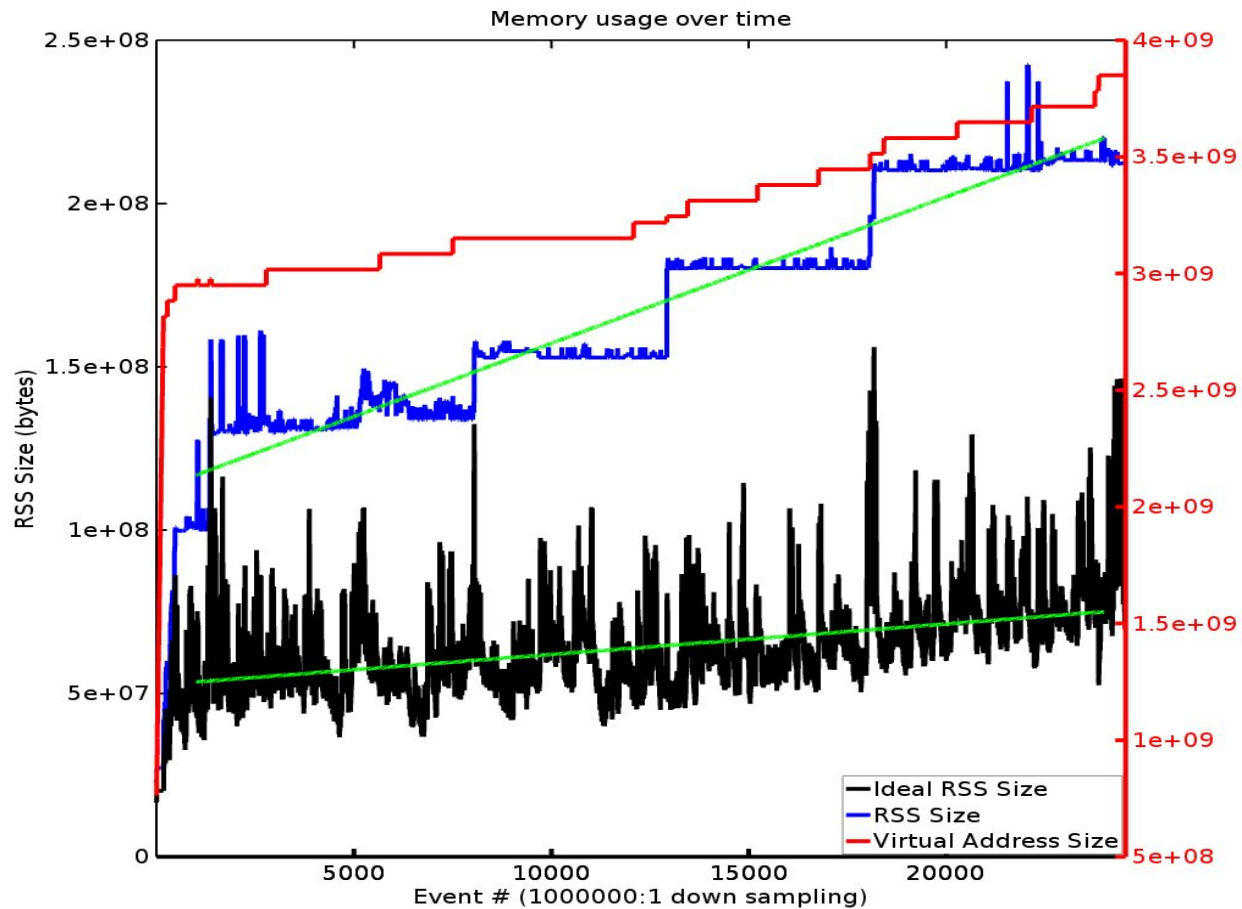
Avg free time: 249 in 12,289,414,779 calls

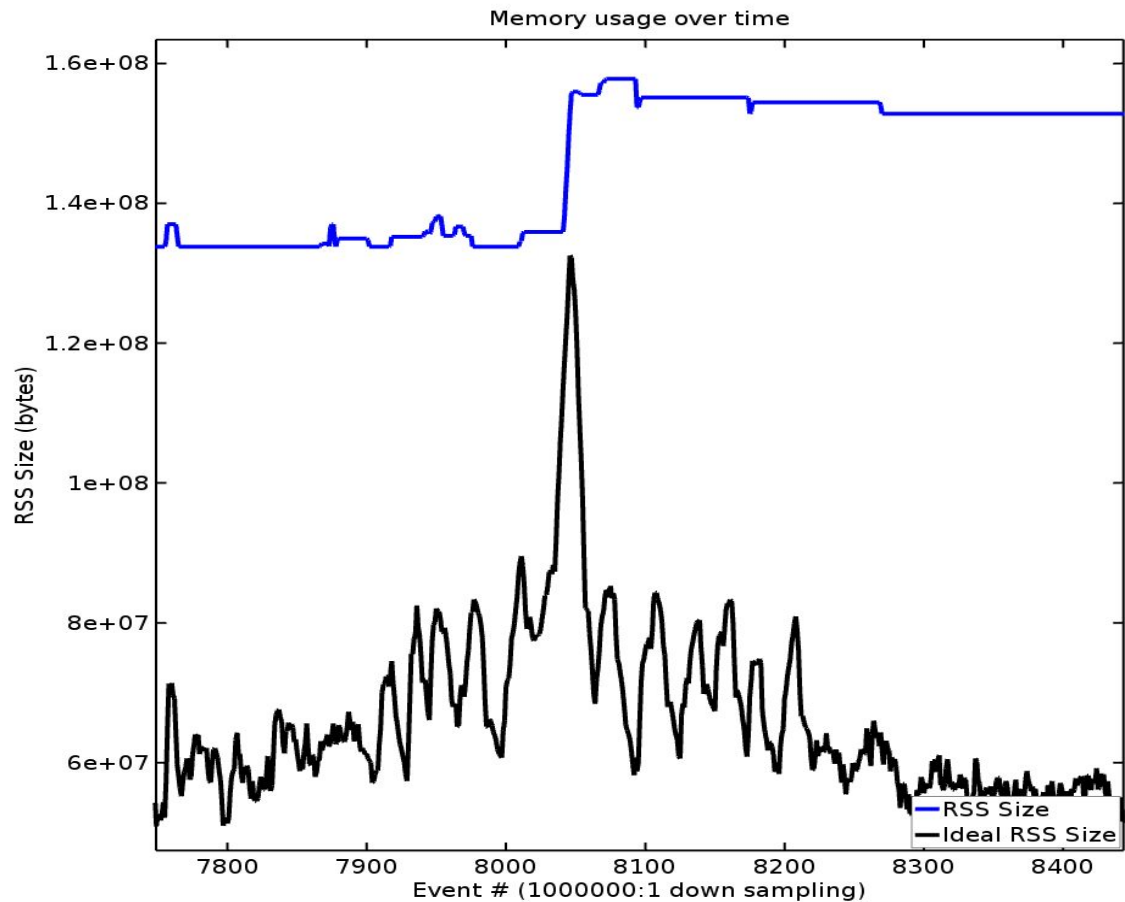
Total call time: 8,077,858,014,177 cycles

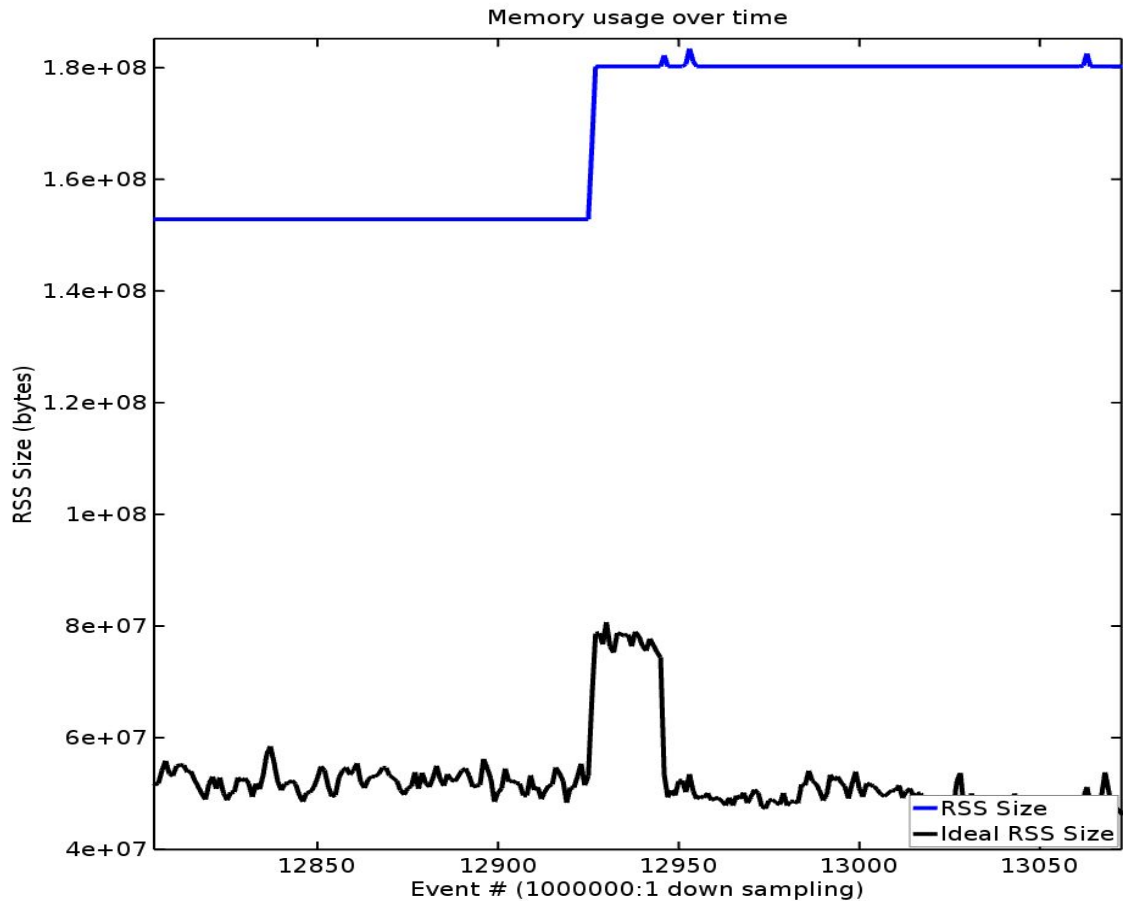
Simulation results

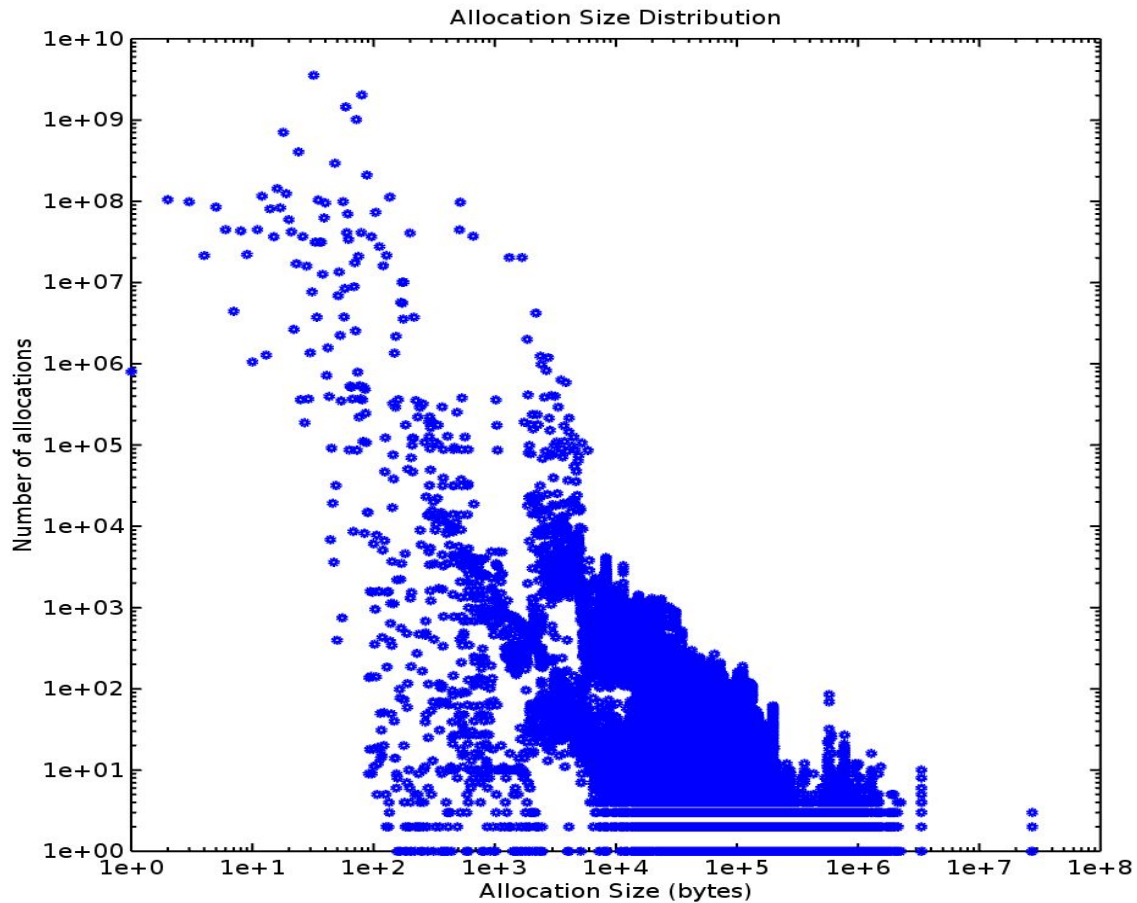
What and how...

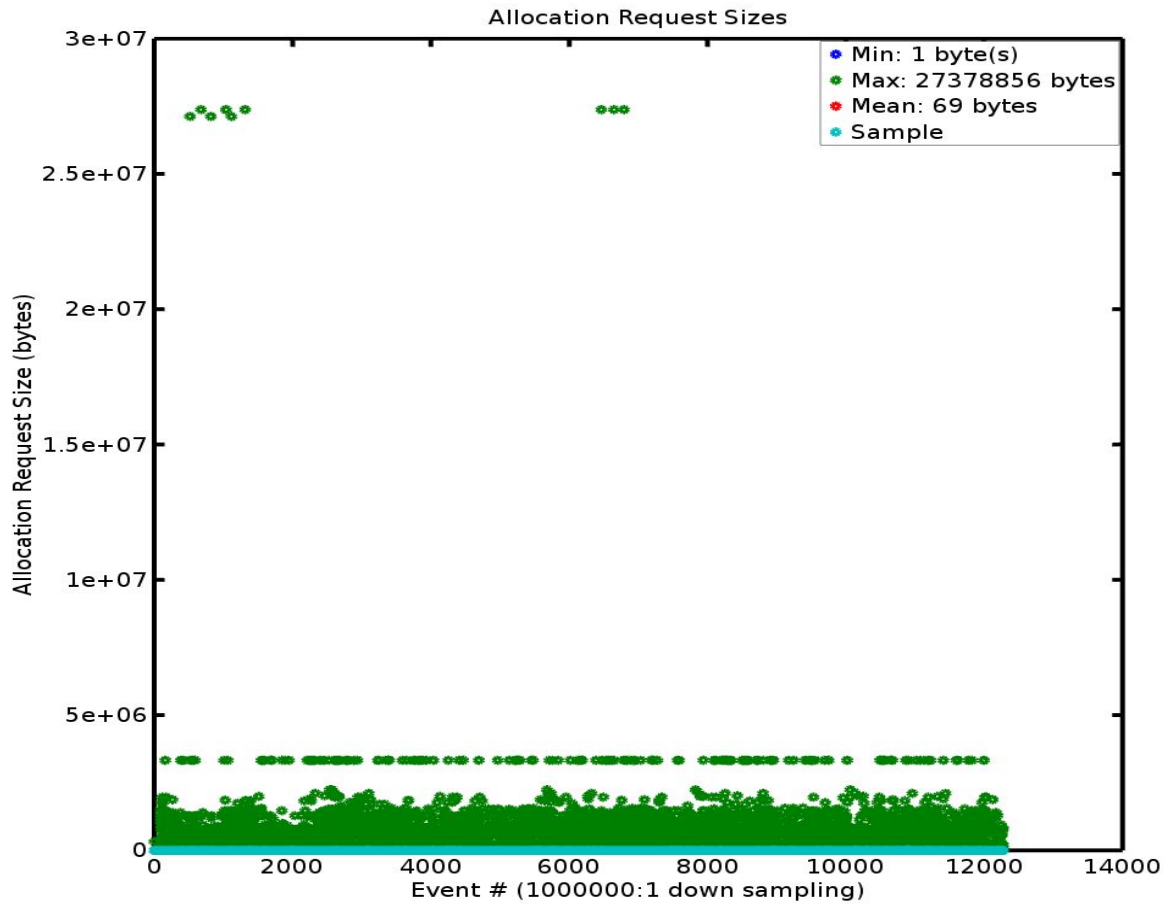
- VmRSS over time (simulator log)
- VmSize over time (simulator log)
- Chunk size over time (data analysis)
- Ideal RSS over time (simulator log)
- A multitude of graphs to look at (data analysis)











Simulation results

What and how...

- Run simulation with different mallocs and evaluate max RSS usage.
- Run simulation with different tunable parameters e.g. `M_MMAP_THRESHOLD`, `M_TRIM_THRESHOLD`, etc.
- Experiment with `malloc_trim()` calling at regular intervals (higher-cost deep trimming).

Simulation results

User feedback.

- Pro: Deeper analysis of allocation patterns.
- Pro: Ability for Red Hat to help easily by providing trace.
- Con: Wish it was on all the time without needing to use an alternate instrumented library.
- Con: Wish it could save results over the network.

Problems in need of solutions

Necessity causes bumps along the way...

- **Input from tracing experts much needed**
- Thread ordering and ownership issues (mremap)
- Lowering the simulator synchronization costs (P&C proofs)
- Lowering the simulator VmSize/VmRSS cost (procfs open/read)
- Condensing trace data (CTF, HDF5)
- Extending to more APIs (LTTng-ust)
- Synchronizing multi-API traces at low cost (global clk, event clk)
- Always-on tracing (dyninst?)

Questions?

- If you ask a question you get a sticker.
- Ask away!

Thank you!

Carlos O'Donnell: carlos@redhat.com

DJ Delorie: dj@redhat.com

Florian Weimer: fweimer@redhat.com