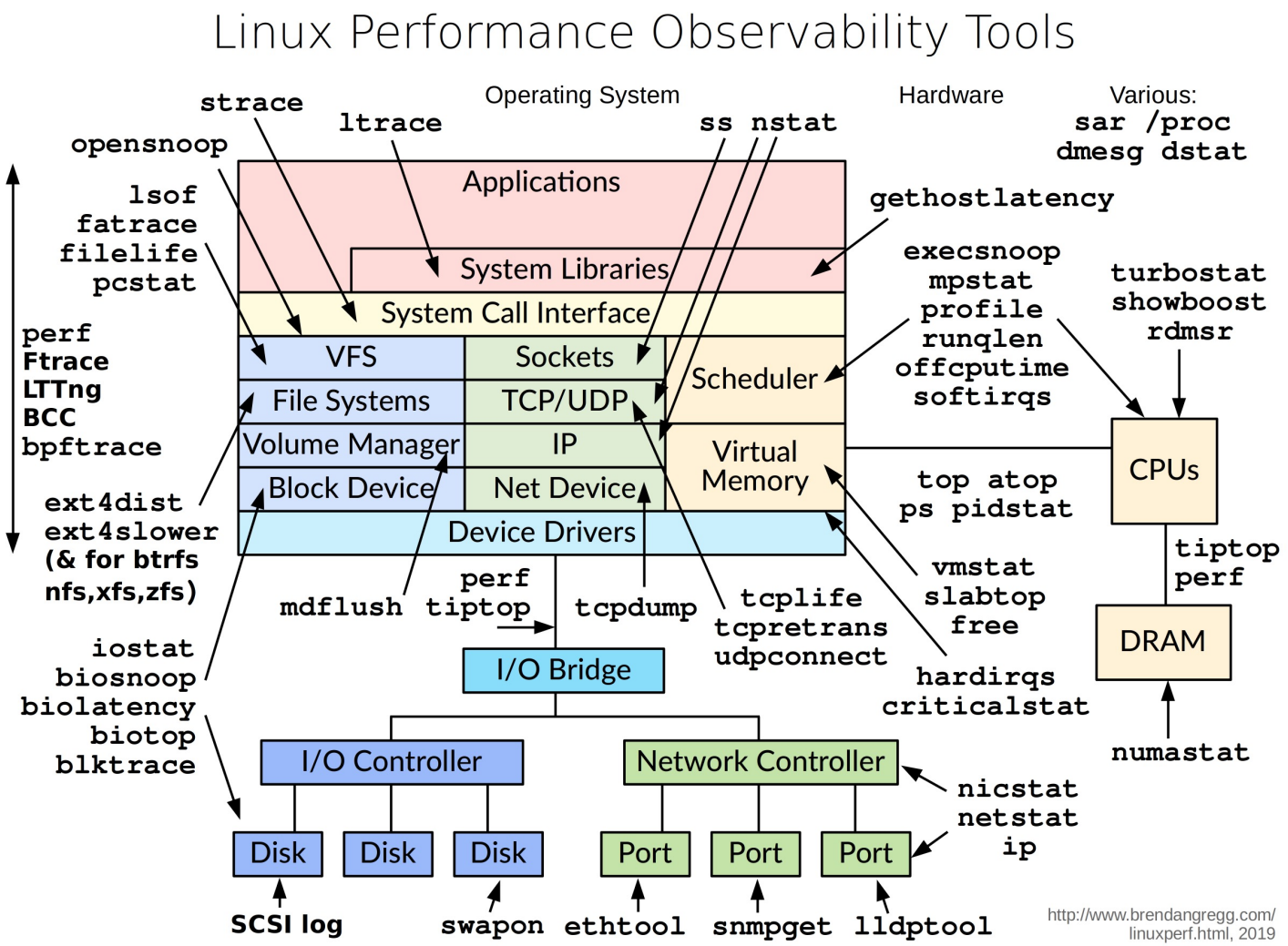
Linux Tracing LTTng

E-Days 2021

NOSER ENGINEERING

Linux Performance Observability Tools

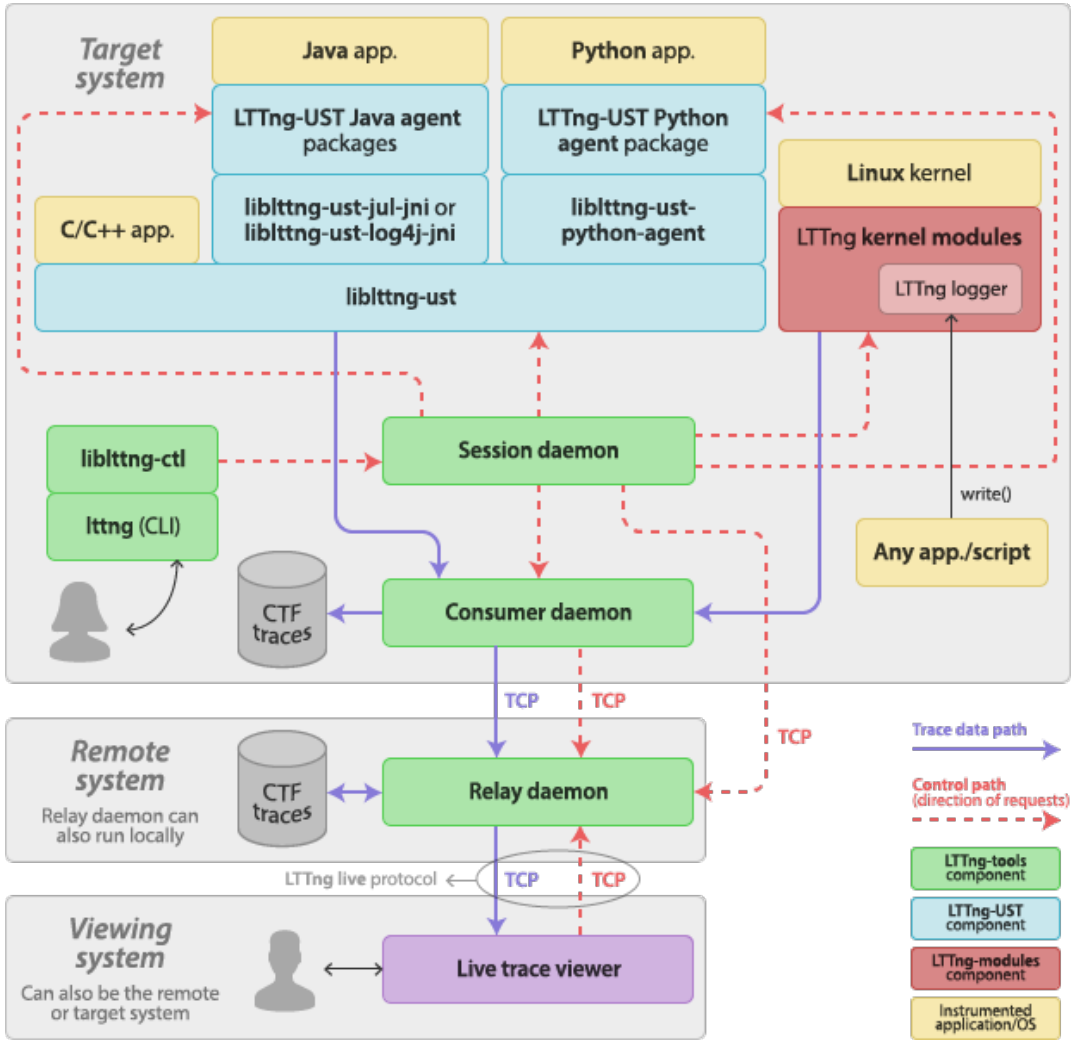


NOSER ENGINEERING

LTTng Advantages

- **Fast kernel tracing** (same speed as ftrace but extracts the syscalls payload)
- **Fast user-space tracing** (does not rely on system calls at every event)
- Native support for C/C++ applications, agents for Java and Python
- Designed to **run continuously in production environments**
- Multi-platform: x86, ARM, PPC, MIPS, s390, Tilera
- Ability to merge kernel and user-space traces
- Multi-host/clock support
- Standard trace format (Common Trace Format aka CTF)
- Packaged by the major distributions including Buildroot and Yocto
- **Vast ecosystem of analysis and post-processing tools.**

LTTng Components



NOSER ENGINEERING

LTTng-tools

- Libraries and command-line interface to control recording sessions:
  - Session daemon (lttng-sessiond(8)).
  - Consumer daemon (lttng-consumerd).
  - Relay daemon (lttng-relayd(8)).
  - Tracing control library (liblttng-ctl).
  - Tracing control command-line tool (lttng(1)).
  - lttng-crash command-line tool (lttng-crash(1)).

LTTng-UST

- Libraries and Java/Python packages to instrument and trace user applications:
  - User space tracing library (liblttng-ust) and its headers to instrument and trace any native user application.

LTTng-UST

- Preloadable user space tracing helpers:
  - liblttng-ust-libc-wrapper (malloc, calloc, realloc, free, memalign, posix_memalign)
  - liblttng-ust-pthread-wrapper (pthread_mutex_lock_req, pthread_mutex_lock_acq, pthread_mutex_trylock, pthread_mutex_unlock)
  - liblttng-ust-cyg-profile (instrument function entry/exit)
  - liblttng-ust-cyg-profile-fast (instrument function entry/exit)
  - liblttng-ust-dl (instrument dynamic linker dlopen, dlclose)

LTTNG-modules

- Linux kernel modules to instrument and trace the kernel:
  - LTTng kernel tracer module.
  - Recording ring buffer kernel modules.
  - Probe kernel modules.
  - LTTng logger kernel module. to instrument Python applications using the standard logging package.

LTTNG-modules required kernel config

- Make sure your target kernel has the following config options enabled:
  - **CONFIG_LTTNG**: for sure….
  - **CONFIG_MODULES**: loadable module support (not strictly required when built into the kernel),
  - **CONFIG_KALLSYMS**: is necessary until the few required missing symbols are exported to GPL modules from mainline
  - **CONFIG_HIGH_RES_TIMERS**: needed for LTTng 2.x clock source,
  - **CONFIG_TRACEPOINTS**: kernel tracepoint instrumentation (enabled as a side-effect of any of the perf/ftrace/blktrace instrumentation features).
  - **CONFIG_KPROBES** (5.7+): use kallsyms for kernel 5.7 and newer.

LTTNG-modules optional kernel config

- The following configuration options will affect the features available from LTTng:
  - **CONFIG_HAVE_SYSCALL_TRACEPOINTS**: system call tracing
  - **CONFIG_PERF_EVENTS**: performance counters
  - **CONFIG_EVENT_TRACING**: needed to allow block layer tracing
  - **CONFIG_KPROBES**: dynamic probes
  - **CONFIG_KRETPROBES**: dynamic function entry/return probes
  - **CONFIG_KALLSYMS_ALL**: state dump of mapping between block device number and name

LTTng Trace Recording Modes

- Tracing to disk with all kernel events enabled can quickly generate huge traces:
  - 54k events/sec on an idle 4-cores laptop, 2.2 MB/sec
  - 2.7M events/sec on a busy 8-cores server, 95 MB/sec
- In addition to filtering and enabling specific events, LTTng offers various recording modes:
  - Local disk and streaming mode
  - Live mode
  - Snapshot mode
  - Rotation mode

Disk and Streaming Modes

- Default mode
- Write buffers to disk or the network when they are full
- Only limited by disk space
- Tracing session needs to be stopped to process the trace
- Use-cases:
    - Understanding the complete life-cycle of a system or an application,
    - Trace exploration (need to identify what is relevant)
    - Post-mortem analyses
    - Reverse engineering
    - Continuous Integration

Live Mode

- Tracing sessions of arbitrary duration and size (same as streaming mode)
- Can attach to a running session and start processing the events while the session is still running
- The trace is still written to disk but we can limit its size with the *tracefile-size* and *tracefile-count* options (on-disk ring buffer)
- Use-cases:
  - Low throughput logging with quick feedback
  - Distributed or embedded systems
  - Continuous monitoring (extracting metrics from events out-of-bound)

Snapshot Mode (Flight Recorder)

- Memory-only tracing (ring-buffer)
- Low overhead while tracing (no I/O)
- On demand, “lttng snapshot record” extracts tracing buffers content from memory to disk or the network. This can also be done via API from within your software.
- Triggers to extract the snapshots can be errors detected by an application, high latencies measured, segmentation faults, time-based sampling, etc,
- The time span covered by a snapshot depends on the buffer size configuration, number of events enabled and the event rate.

Snapshot Mode (Flight Recorder)

- Use-cases:
  - Fault investigation: get the full activity a few seconds before an error or high latency occurred
  - Profiling: get a sense of the machine activity periodically
  - When a Continuous Integration worker detects an error

Rotation Mode

- Archive a tracing session’s current chunk
- Allows to process/archive/delete/compress a chunk of a trace while it is still writing in a separate directory,
- The trace can run indefinitely but the chunks can be processed like offline traces (disk or streaming mode)
- Timer-based or size-based auto-rotation available

Rotation Mode

- Use-cases:
  - Continuous monitoring: periodically rotate and extract/plot low-level metrics from the trace
  - Smaller traces to process than with the default mode
  - Spreading the post-processing load (send chunks for analysis to available worker servers)
  - Archiving/Compression

Trace Compass

- ***Trace Compass*** is an open source application to solve performance and reliability issues by reading and analyzing traces and logs of a system.
- Its goal is to provide views, graphs, metrics, and more to help extract useful information from traces, in a way that is more user-friendly and informative than huge text dumps.
- Get it from https://www.eclipse.org/tracecompass/

Trace Compass Key Features

Offline analysis of complex issues

- Real-time deadline investigation
- Latency analysis
- Log correlation with operating system traces
- Network packet correlation across layers
- Identification of relevant information in large amounts of trace data
- Critical path analysis
- Causes of high processor usage and memory leaks
- Correlation of hardware and software components execution traces
- Time synchronization of traces from different nodes
- Symbol name resolution using debug information
- Additional analyses available with the Trace Compass Incubator
- and more!

Trace Compass Key Features

Multiple trace formats supported

- Common Trace Format (CTF), including but not limited to:
  - Linux LTTng kernel traces
  - Linux LTTng-UST userspace traces
  - Linux Perf traces converted to CTF
  - Bare metal traces
  - Integration with the LTTng-Analyses scripts
- Hardware traces (e.g. IEEE Nexus 5001 CTF conversion). See also this link.
- GDB traces for debugging
- The Best Trace Format (BTF) for OSEK
- The libpcap (Packet CAPture) format, for network traces
- Custom text or XML parsers that can be added right from the graphical interface by the user
- Can be extended to support various log or trace files.
- Provided by the TraceCompass Incubator:
  - Linux FTrace raw textual format
  - Google's trace event json format
  - Additional Linux Perf2ctf traces features
  - Android traces
  - UFtrace format

Trace Compass Benefits

- Reduce time to identify faults
- Observe multi-core, heterogeneous, virtualized, and distributed systems
- Use the same analysis tool for development, testing, and production
- Extend the framework to fit the needs of your organization
- Avoid vendor lock-in by using an open source solution

LTTng-UST Function Entry/Exit

- Example of function tracing by using preloaded liblttng-ust-cyg-profile.so

Don’t forget: Compile with --finstrument-functions

LTTng-UST Pre-Build Helpers

- Example of tracing by using all of the LTTng-UST pre-build helpers

LTTng-UST tracef & tracelog

- Example of printf like traces

LTTng-UST Custom Trace Point

- Example of creating a custom trace point

LTTng-UST Dynamic Tracing with dtrace

- Example of dynamic tracing with dtrace

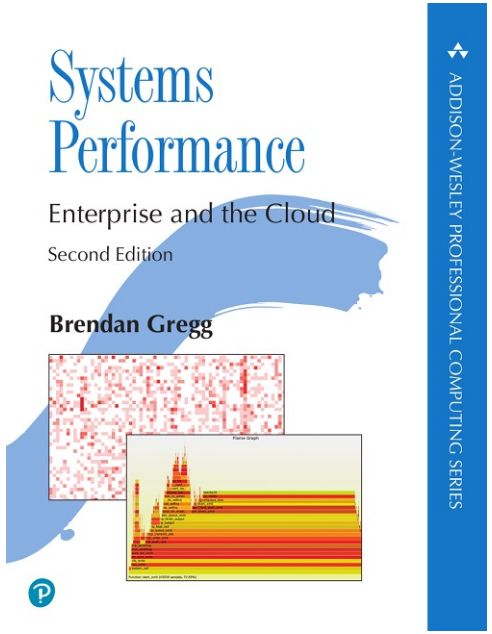
Cold Case ThermalZone

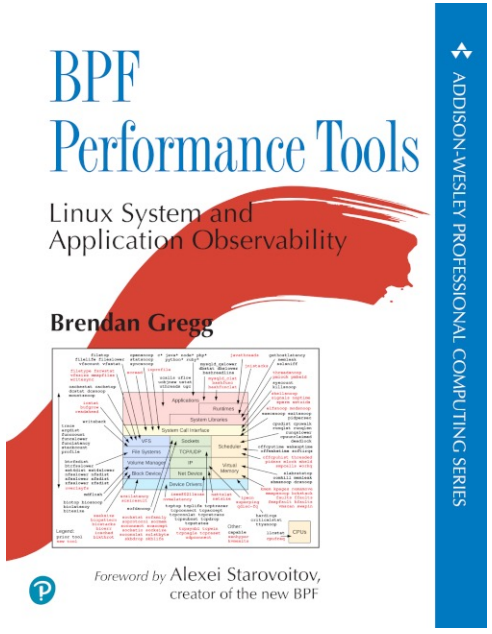
- Scenario
  - Measure interprocess communication latency between a producer and a consumer process
  - Messages are send with protobuf encapsulation via linux message queues between the two processes
  - Uses LTTng Kernel & UST tracing
  - Uses LTTng snapshot mode, snapshot triggered if latency if over a given threshold
- Task:
  - Hunt for latency outliers

Links LTTng and tracing in general

- https://lttng.org/docs
- https://brendangregg.com

Books





- https://brendangregg.com/systems-performance-2nd-edition-book.html
- https://brendangregg.com/bpf-performance-tools-book.html

NOSER ENGINEERING

Slides, examples & more

- https://github.com/schenkmi/Noser-E-Days-2021-LTTng-Tracing