RTLA

Real-time Linux Analysis toolset

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

SEMINAR SERIES

Daniel Bristot de Oliveira
Senior Principal Engineer,
Red Hat

Real-time Linux

- Linux has been used as an RTOS - it is a fact!
- There are multiple reasons for people to use it
  - Software stack and availability
  - Man-power
- But also because Linux achieves the desired timing behavior
- Some key features to help on that are:
  - The fully preemptive mode
  - Real-time scheduling
    - SCHED_DEADLINE

Real-time Linux testing

- One of the problems, however, is the way that we show the timing properties of Linux
- Nowadays, Linux is tested using **blackbox tools** that mimic typical workload:
    - **Event** driven application: **cyclictest**
    - **Polling** like application: **sysjitter/oslat**
- They report a "**latency**", and this is important for many use-cases. For example:
    - The kernel-rt has to deliver < 150 us cyclictest latency under stress
    - cyclictest latency of 10~20 us on isolated & tuned systems

Real-time Linux testing

- **The blackbox approach works, but it has some drawbacks**
  - It gives no root cause analysis
- **The root cause analysis is generally done using tracing**
  - But tracing is not that accessible for non-experts
- **Real-time to the masses**
  - **All kernel developers will have to run RT analysis**
  - But not all are interested in learning all the details

RTLA: A new approach

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

SEMINAR SERIES

Real-time Linux Approach

- **RTLA follows a white-box approach**
- It integrates the workload and tracing
- In kernel:
  - Integrated tracer and workload
- In user-space
  - Easy to use interface
  - Data analysis

RTLA: kernel tracers

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

SEMINAR SERIES

Kernel tracers

- **RTLA uses two kernel tracers**
- **osnoise tracer**
  - Measures the Operating System noise/interference from high prio tasks
  - IOW: sysjitter/oslat on steroids
- **timerlat tracer**
  - Measures the activation delay of a timer triggered task
  - IOW: cyclictest on steroids

osnoise tracer

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

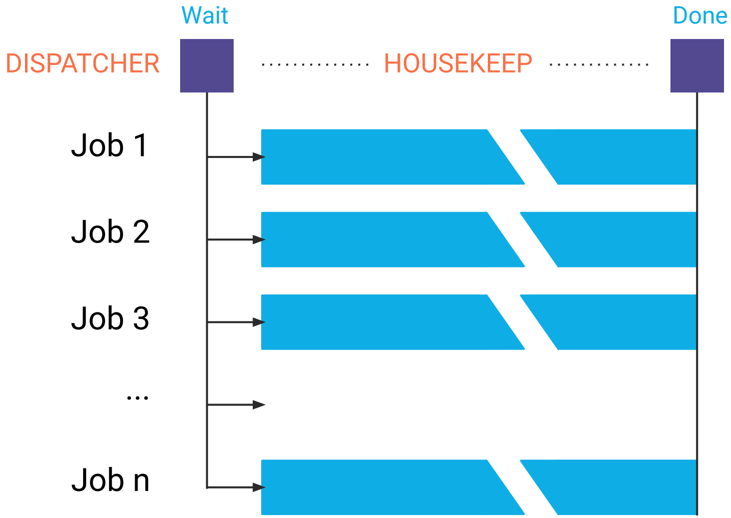
SEMINAR SERIES

Operating system noise

- The Operating System Noise (**OS Noise**) is a well defined High Performance Computing (**HPC**) metric
- It is the amount of **interference** experienced by an application due to **operating system activities**
- It is generally a fine grained metric

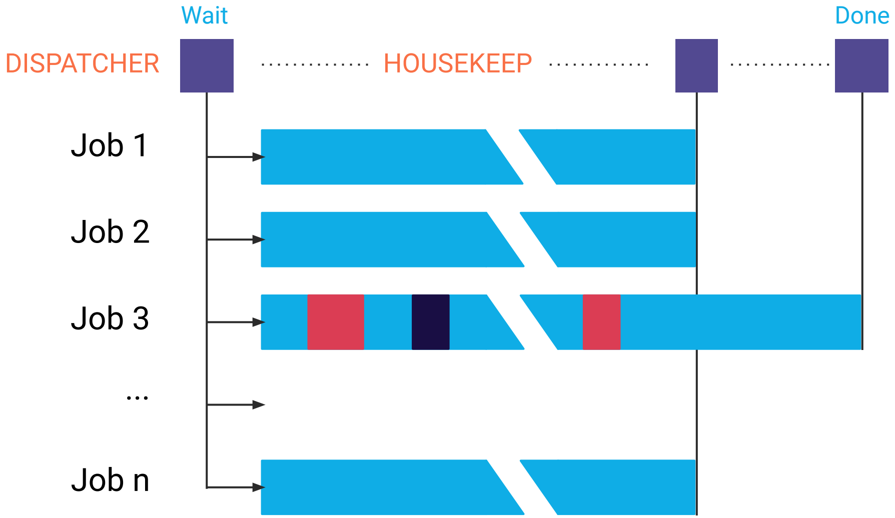
Operating System noise

- Generally, HPC workloads are composed of **parallel jobs**
- The system is configured with **CPUs dedicated** to the jobs
- A dispatcher launches jobs to these CPUS and waits for completion



Operating System noise

- The side effects of the OS Noise to the workload can influence the total response time of the system.
    - Both in parallel and pipeline workloads
- Some critical HPC RT workloads requires OS Noise to be less than 20 us.



OS Noise tracer

- **osnoise** is a kernel tracer that also dispatches the workload
  - The workload runs in the kernel
- **It mimics HPC workload**
  - One thread per CPU
  - Detects noise by computing the delta between two consecutive reads of the time
- It has integrated tracing events to identify the source of the noise
  - In kernel lockless synchronization -> no false positives
- It detects high priority tasks that interfere the osnoise workload
  - osnoise can also detect hw/vm induced latency

OS Noise tracer and safety critical systems

- It is common practice to **partition the system** in critical and non-critical domains
  - **Isolated/dedicated CPUs for critical workload**
    - Or even to a single workload or a middleware/framework
- The osnoise tracer is useful to:
  - **assess the partitioning/isolation**
  - identify how much interference the OS is adding to the critical load
    - Causing delay in the response time of critical workload

Using the osnoise tracer

```
[root@f32 ~]# cd /sys/kernel/tracing/
[root@f32 tracing]# echo osnoise > current_tracer
[root@f32 tracing]# cat trace
# tracer: osnoise
#
#                                 _-----=> irqs-off
#                                / _----=> need-resched
#                              | / _---=> hardirq/softirq
#                              || / _--=> preempt-depth                            MAX
#                              || /                                             SINGLE     Interference counters:
#                              ||||               RUNTIME      NOISE   % OF CPU  NOISE    +-----------------------------+
#           TASK-PID      CPU# ||||   TIMESTAMP    IN US       IN US  AVAILABLE  IN US     HW    NMI    IRQ   SIRQ THREAD
#              | |         |   ||||      |           |             |    |            |      |      |      |      |      |
           <...>-859     [000] ....    81.637220: 1000000        190  99.98100       9     18      0   1007     18      1
           <...>-860     [001] ....    81.638154: 1000000        656  99.93440      74     23      0   1006     16      3
           <...>-861     [002] ....    81.638193: 1000000       5675  99.43250     202      6      0   1013     25     21
           <...>-862     [003] ....    81.638242: 1000000        125  99.98750      45      1      0   1011     23      0
           <...>-863     [004] ....    81.638260: 1000000       1721  99.82790     168      7      0   1002     49     41
           <...>-864     [005] ....    81.638286: 1000000        263  99.97370      57      6      0   1006     26      2
           <...>-865     [006] ....    81.638302: 1000000        109  99.98910      21      3      0   1006     18      1
           <...>-866     [007] ....    81.638326: 1000000       7816  99.21840     107      8      0   1016     39     19
```

OS Noise tracer options

- Configuration files inside `/sys/kernel/trace/osnoise`
  - `cpus`: CPUs at which an osnoise thread will execute.
  - `period_us`: the period of the osnoise thread.
  - `runtime_us`: how long an osnoise thread will look for noise in the period
  - `stop_tracing_us`: stop system tracing if a single noise is >= than set here
  - `Stop_tracing_total_us`: stop system tracing if total noise is >= than set here

- `/sys/kernel/trace/tracing_threshold`
  - The minimum delta between two time() reads to be considered as noise, in us.
  - When set to 0, the default value will will be used, which is currently 5 us.

osnoise analysis

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

SEMINAR SERIES

What can cause OS Noise?

- Any sort of task tha interference (preempt) the osnoise workload
- Linux task abstractions:
  - **NMI**
  - **IRQs**
  - **Softirqs**
  - **Threads**
- But also the hardware can interfere
  - **SMIs**
  - **VMs**

osnoise tracepoints

- One tracepoint for each task abstraction:
  - osnoise:**nmi**_noise
  - osnoise:**irq**_noise
  - osnoise:**softirq**_noise
  - osnoise:**thread**_noise
- They report the amount of noise
  - The values are free from nested interference
    - e.g., a thread_noise noise is free from any IRQ/Softirq/NMI interference that it could face
- osnoise:sample_threshold: the total noise observed by the workload

Using osnoise tracepoints & root cause

```
[root@f32 ~]# cd /sys/kernel/tracing/
[root@f32 tracing]# echo osnoise > current_tracer
[root@f32 tracing]# echo osnoise > set_event
[root@f32 tracing]# echo 8 > osnoise/stop_tracing_us
[root@f32 tracing]# cat trace
[...]
  osnoise/8-960     [007] d.h.  5789.857530: irq_noise: local_timer:236 start 5789.857527123 duration 1867 ns
  osnoise/8-961     [008] d.h.  5789.857532: irq_noise: local_timer:236 start 5789.857529929 duration 1845 ns
  osnoise/8-961     [008] dNh.  5789.858408: irq_noise: local_timer:236 start 5789.858404871 duration 2848 ns
migration/8-54      [008] d...  5789.858413: thread_noise: migration/8:54 start 5789.858409300 duration 3068 ns
  osnoise/8-961     [008] ....  5789.858413: sample_threshold: start 5789.858404555 duration 8812 ns interferences 2
```

timerlat tracer

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

SEMINAR SERIES

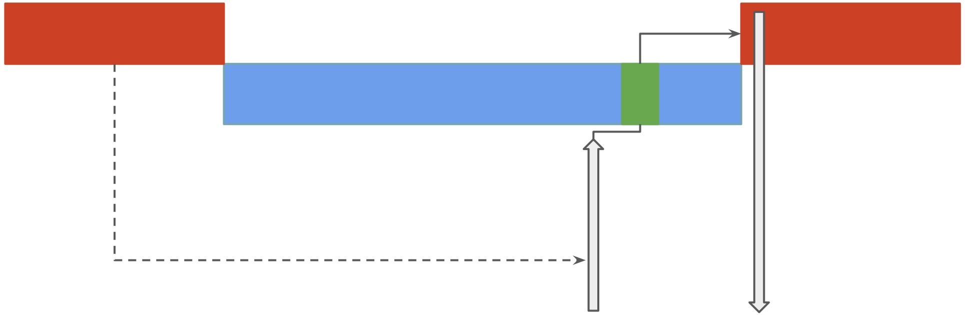
Timer latency

- Timer latency has been used as a metric by the real-time Linux kernel developers
  - cyclictest is indeed a timer testing tool
- It empirically measures the observed **scheduling latency** of the highest priority thread - or a thread at any priority
- **timerlat tracer** measure the same metric, but it is integrated with tracing.

Timer latency and safety critical systems

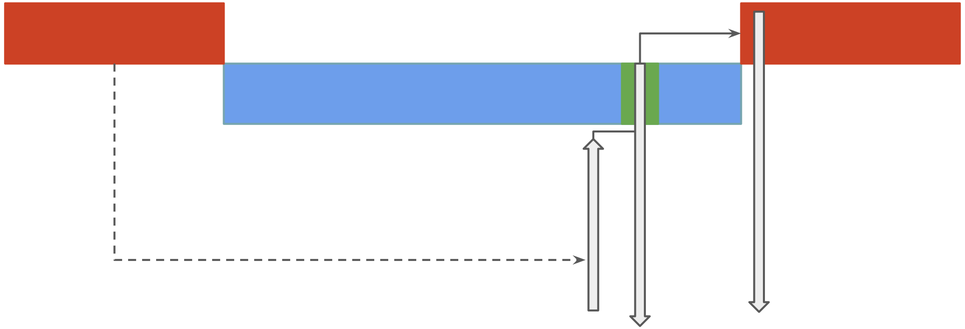
- Measuring the timer latency is equivalent to measure the response to an external event
    - For timer trigger events
    - Or any like any interrupt from hardware
- The timerlat tracer is useful:
    - To assess that externally triggered events are timely handled.
    - To identify how much activation latency non-critical load is adding to the critical load
        - Causing delay in the response time of critical workload

Task activation delay



Set a timer a *t* . . . . *t* . . . . . .Δ = t' – t

Task activation delay



Set a timer a *t* . . . . *t* . *t'* . . . *t''* -> $\Delta' = t' - t$

$\Delta'' = t'' - t$

Using the timerlat tracer

```
[root@f32 ~]# cd /sys/kernel/tracing/
[root@f32 tracing]# echo timerlat > current_tracer
[root@f32 tracing]# cat trace
# tracer: timerlat
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            || /
#                            ||||             ACTIVATION
#         TASK-PID      CPU# ||||   TIMESTAMP    ID            CONTEXT                LATENCY
#            | |         |   ||||      |         |                  |                       |
        <idle>-0       [000] d.h1    54.029328: #1     context    irq timer_latency       932 ns
         <...>-867     [000] ....    54.029339: #1     context thread timer_latency     11700 ns
        <idle>-0       [001] dNh1    54.029346: #1     context    irq timer_latency      2833 ns
         <...>-868     [001] ....    54.029353: #1     context thread timer_latency      9820 ns
        <idle>-0       [000] d.h1    54.030328: #2     context    irq timer_latency       769 ns
         <...>-867     [000] ....    54.030330: #2     context thread timer_latency      3070 ns
        <idle>-0       [001] d.h1    54.030344: #2     context    irq timer_latency       935 ns
         <...>-868     [001] ....    54.030347: #2     context thread timer_latency      4351 ns
```

ENABLING LINUX IN SAFETY APPLICATIONS

Timerlat tracer options

- Configuration files inside `/sys/kernel/trace/osnoise`
  - `cpus`: CPUs at which a timerlat thread will execute.
  - `period_us`: the timer period
  - `stop_tracing_us`: stop the system tracing if IRQ latency>= than set here
  - `stop_tracing_total_us`: stop the system tracing if thread latency is >= than set here
  - `print_stack`: save the IRQ stack trace to print in case of latency >= than set

timerlat analysis

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

SEMINAR SERIES

What can cause timer latency?

- Linux task abstractions:
  - NMI
  - IRQs
  - softirqs
  - Higher priority thread
- Previously running thread with **preemption || irq** disabled

osnoise tracepoints

- One tracepoint for each task abstraction:
  - osnoise:**nmi**_noise
  - osnoise:**irq**_noise
  - osnoise:**softirq**_noise
  - osnoise:**thread**_noise
- They report the amount of noise
- softirq and thead noise account from the timer IRQ handler
  - **So it measures the noise actually added to timer thread**

Using the timerlat tracer

```
[root@f32 ~]# cd /sys/kernel/tracing/
[root@f32 tracing]# echo timerlat > current_tracer
[root@f32 tracing]# echo 1 > events/osnoise/enable
[root@f32 tracing]# echo 500 > osnoise/stop_tracing_total_us
[root@f32 tracing]# echo 500 > osnoise/print_stack
[root@f32 tracing]# tail -21 per_cpu/cpu7/trace
        insmod-1026    [007] dN.h1..   200.201948: irq_noise: local_timer:236 start 200.201939376 duration 7872 ns
        insmod-1026    [007] d..h1..   200.202587: #29800 context    irq timer_latency      1616 ns
        insmod-1026    [007] dN.h2..   200.202598: irq_noise: local_timer:236 start 200.202586162 duration 11855 ns
        insmod-1026    [007] dN.h3..   200.202947: irq_noise: local_timer:236 start 200.202939174 duration 7318 ns
        insmod-1026    [007] d...3..   200.203444: thread_noise:   insmod:1026 start 200.202586933 duration 838681 ns
    timerlat/7-1001    [007] .......   200.203445: #29800 context thread timer_latency    859978 ns
    timerlat/7-1001    [007] ....1..   200.203446: <stack trace>
=> timerlat_irq
=> __hrtimer_run_queues
=> hrtimer_interrupt
=> __sysvec_apic_timer_interrupt
[...continue...]
```

Using the timerlat tracer

```
[...]
        insmod-1026    [007] d..h1..   200.202587: #29800 context    irq timer_latency      1616 ns
        insmod-1026    [007] dN.h2..   200.202598: irq_noise: local_timer:236 start 200.202586162 duration 11855 ns
        insmod-1026    [007] dN.h3..   200.202947: irq_noise: local_timer:236 start 200.202939174 duration 7318 ns
        insmod-1026    [007] d...3..   200.203444: thread_noise:   insmod:1026 start 200.202586933 duration 838681 ns
    timerlat/7-1001    [007] .......   200.203445: #29800 context thread timer_latency    859978 ns
    timerlat/7-1001    [007] ....1..   200.203446: <stack trace>
=> timerlat_irq
=> __hrtimer_run_queues
=> hrtimer_interrupt
=> __sysvec_apic_timer_interrupt
=> asm_call_irq_on_stack
=> sysvec_apic_timer_interrupt
=> asm_sysvec_apic_timer_interrupt
=> delay_tsc
=> dummy_load_1ms_pd_init
=> do_one_initcall
=> do_init_module
=> __do_sys_finit_module
=> do_syscall_64
=> entry_SYSCALL_64_after_hwframe
```

RTLA

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

SEMINAR SERIES

Real-time Linux Analysis

- rtla is a **user-space** tool that serves as **front-end** for **setup**, **tracing** and **data analysis**
- **It transforms the tracers into a benchmark tool**
- It is in **C**, hosted inside the **tools/tracing/rtla** in the **kernel repo**
- Two tools in the initial implementation:
  - **rtla osnoise**: measures the operating system noise
  - **rtla timerlat**: measures the timer latency

rtla osnoise

- **rtla osnoise** is an interface to **osnoise tracer**
  - Adds more options to the tracer
    - e.g., setting priority to threads
  - Interface for other tracing features like tracepoints and histograms
- Two different modes:
  - **osnoise top**: shows an interactive view of the osnoise summary output
  - **osnoise hist**: shows a histogram of the osnoise sample tracepoint

rtla timerlat

- **rtla timerlat** is an interface to **timerlat tracer**
  - Adds more options to the tracer
    - e.g., setting priority to threads
  - Interface for other tracing features like tracepoints and histograms
- Two different modes:
  - **timerlat top**: shows an interactive view of the osnoise summary output
  - **timerlat hist**: shows a histogram of the osnoise sample tracepoint

rtla timerlat: how easy it is?

- I am a **user** testing my **kernel-rt** setup, and I want to **measure** the latency and generate a **report** if my **latency is higher than 50 us**?
- Nowadays, this requires:
  - Using **cyclictest** with stop tracing
  - **Instructions about setting tracing** (asking in IRC or mailing list?)
  - Figuring things out from tracing, computing execution time **by hand**/scripts.
- **How much easier is my life using rtla?**

rtla timerlat: how easy it is?

- **timerlat top -a 50**

rtla timerlat: how easy it is?

- **timerlat top -a 50**
- It measures latency
- Sets up a tracing session
- Enables the minimum required trace events
  - osnoise: events
  - stacktrace for the IRQ handler
- Stops the trace if a 50 us latency is hit, saving the result to a timerlat_trace.txt

RTLA is the automation of an expert analysis

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

SEMINAR SERIES

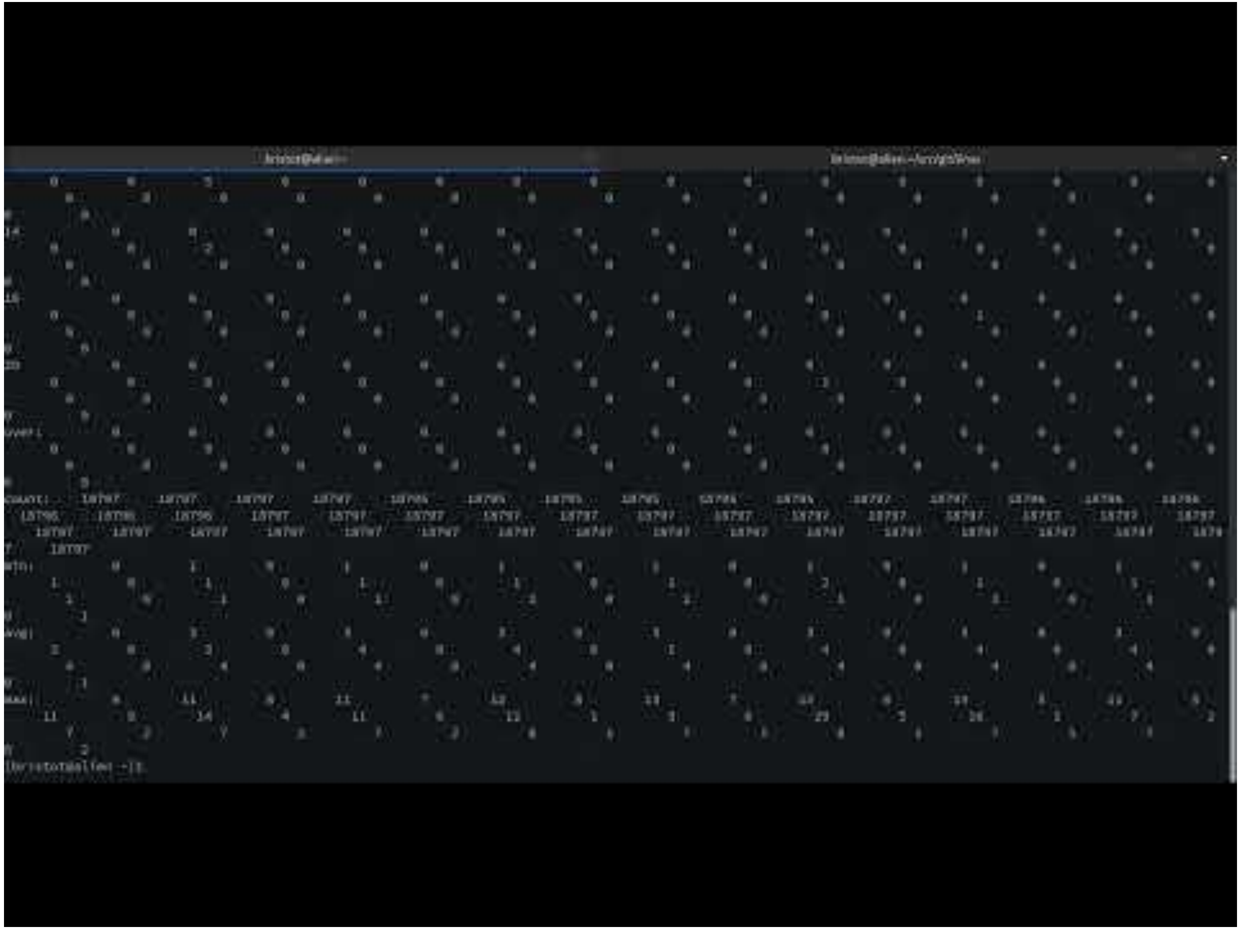
RTLA demo

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

SEMINAR SERIES

Demo:



https://www.youtube.com/watch?v=3sGM076mLRQ&t=1s

ENABLING LINUX IN SAFETY APPLICATIONS

RTLA status

- **RTLA is upstream!**
  - **Tracers since 5.14**
  - **RTLA since 5.17**
  - Advanced trace support queued for 5.18
- Tracers enabled on Fedora/CentOS/Red Hat
- RTLA package on the way to Fedora/CentOS/Red Hat
- More tools and analysis are on the way
  - rtsl is next -> https://bristot.me/demystifying-the-real-time-linux-latency/

Thanks

ELISA

ENABLING LINUX IN SAFETY APPLICATIONS

SEMINAR SERIES