Did you behave in real-time? A probabilistic evaluation with trace data

Benno Bielmeier[1], **Wolfgang Mauerer**[1,2], **Ralf Ramsauer**[1]

LPC 2024 / Real-Time MC
September 19, 2024

[1]Technical University of Applied Scienes Regensburg [2]Siemens AG, Technology

RT Behaviour

Timing Analysis (TA)

- Static TA

Timing Analysis (TA)

- Static TA
- Measurement-Based TA

Image source: osadl.org

RT Behaviour

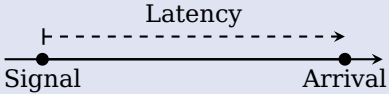
Timing Analysis (TA)

- Static TA
- Measurement-Based TA
- Measurement-Based Probabilistic TA

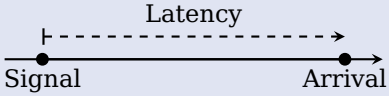
Quality vs. Quantity

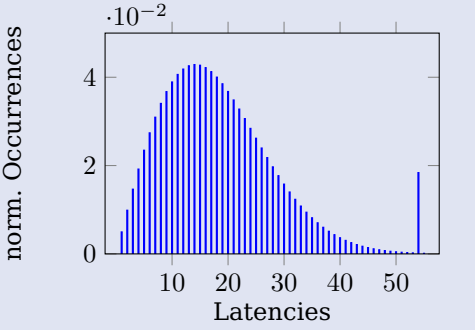
Confidence vs. Measurement Period

Measurement Approach

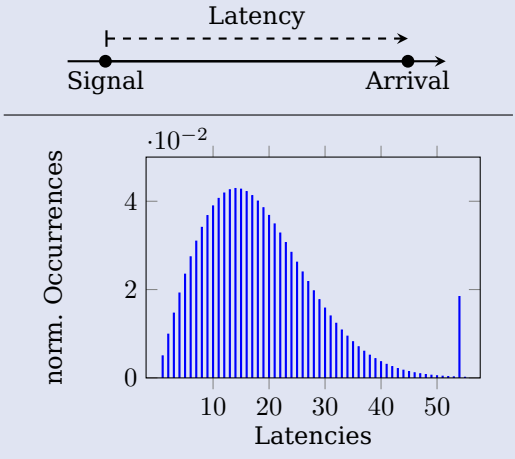


Measurement Approach

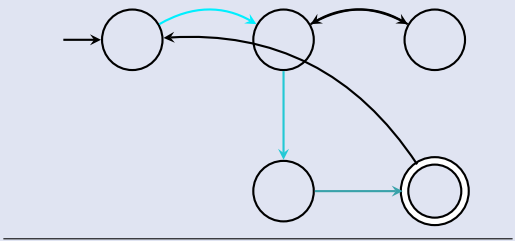




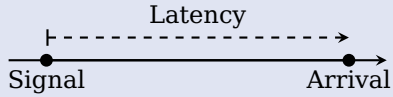
Measurement Approach

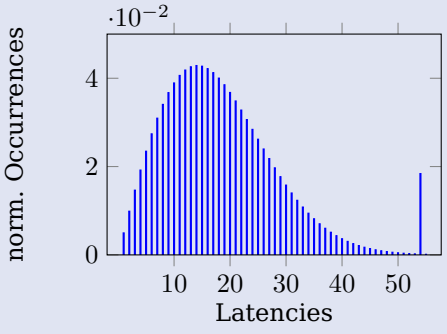


Hybrid Approach

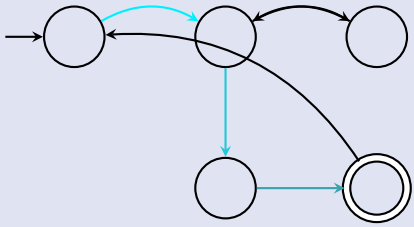


Measurement Approach



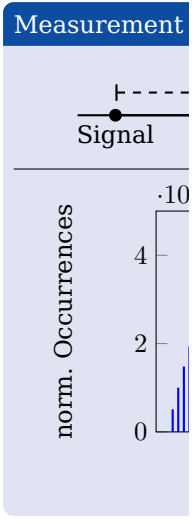


Hybrid Approach



$$SMM = (S, P, F_{ij}(t), \pi)$$

$$\mathbb{E}[F(i, j_f)] = \mathbb{E}[T_i] + \sum_{k \neq j_f} P_{ik} \mathbb{E}[F(k, j_f)]$$



Modeling the Behavior of Threads in the PREEMPT_RT Linux Kernel Using Automata

Daniel Bristot de Oliveira
Red Hat, Inc.
Scuola Superiore Sant'Anna
Universidade Federal de Santa Catarina
bristot@redhat.com

Tommaso Cucinotta
Scuola Superiore Sant'Anna
tommaso.cucinotta@santannapisa.it

Rômulo Silva de Oliveira
Universidade Federal de Santa Catarina
romulo.deoliveira@ufsc.br

ABSTRACT

This article proposes an automata-based model for describing and verifying the behavior of thread management code in the Linux PREEMPT_RT kernel, on a single-core system. The automata model defines the events that influence the timing behavior of the execution of threads, and the relations among them. This article also presents the extension of the Linux trace features that enable the trace of such events in a real system. Finally, one example is presented of how the presented model and tracing tool helped catching an inefficiency bug in the scheduler code and ultimately led to improving the kernel.

CCS CONCEPTS

•**Computer systems organization** → **Real-time operating systems**; *Embedded systems*;

KEY WORDS

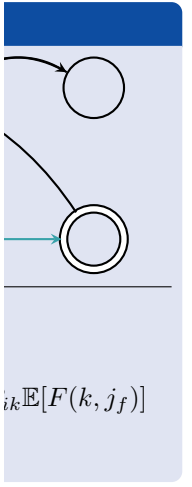
Real-time systems, Linux kernel, behavioral modeling, code verification, automata, tracing.

1 INTRODUCTION

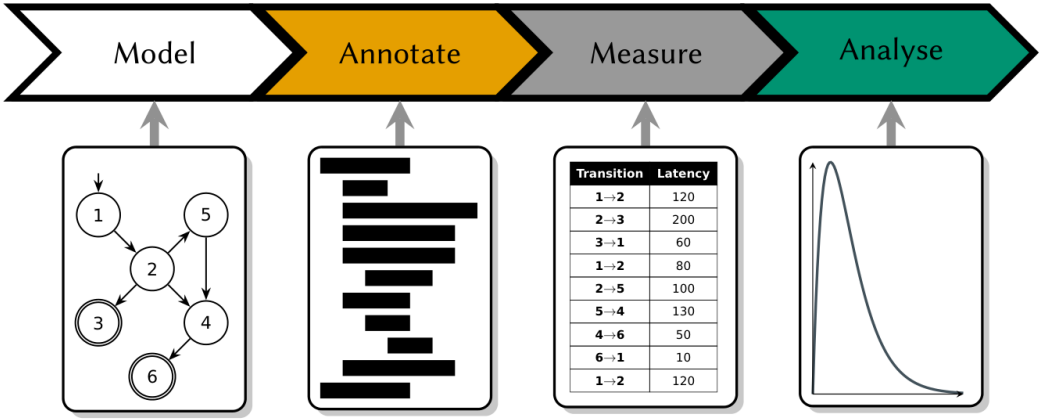
Real-time Linux has been a research topic for more than a decade now, with many scheduling algorithms being implemented in Linux [4, 7, 24, 32]. Despite the constant interaction of Linux development

Then, interrupts must be disabled as well, to avoid race conditions with interrupt handlers. Hence, delays in the scheduling and interrupt handler are created during activation of a thread [12].

The understanding of such operations, and how they affect the timing behavior of a task, are fundamental for the development of real-time algorithms for Linux. However, the amount of effort required for a researcher to understand all these constraints is not negligible. Rather, it might take years for a newcomer to understand the internals of the Linux kernel. The complexity of Linux is indeed a barrier, not only for researchers but for developers as well. Inside the kernel, scheduling operations interact with low-level details of the underlying processor and memory architectures, where complex locking protocols and "hacks" are used. This is done to ensure that such a general-purpose operating system (GPOS) behaves as efficiently as possible in the average case, while at the same time it is done to ensure that, with proper tuning, the kernel can serve an increasing number of real-time use cases as well, turning effectively into a real-time operating system (RTOS). The progressive chasing and elimination over the years of any use of the old global kernel lock, the extensive use of fine-grain locking, the widespread adoption of memory barrier primitives, or even the post-ponement of most interrupt handling code to kernel threads as done in the PREEMPT_RT kernel, are all examples of a big commitment into reducing the duration of non-preemptible kernel sections to the bare minimum, while allowing for a greater control over the priority and scheduling among the various in-



Idea & Approach



Measurement-based Timing Analysis

Assumptions and Requirements

- System/Domain expertise required
- Payload-specific
- Precise timing information of the events

Limitations

- Measurements influence system behaviour
- Extreme outliers may remain unseen

Measurement-based Probabilistic Timing Analysis

Assumptions and Requirements

- System/Domain expertise required
- Payload-specific
- Precise timing information of the events

Limitations

- Measurements influence system behaviour
- Extreme outliers may remain unseen

Measurement-based Probabilistic Timing Analysis

Assumptions and Requirements

- System/Domain expertise required
- Payload-specific
- Precise timing information of the events

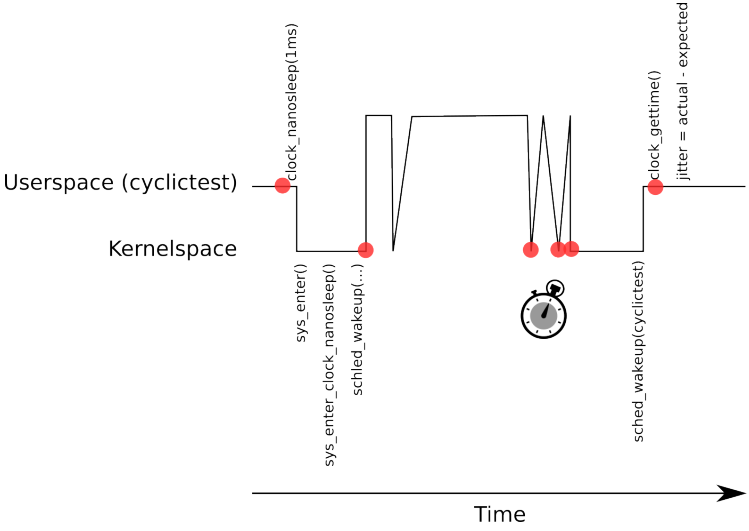
Limitations

- Measurements influence system behaviour
- Extreme outliers may remain unseen

Opportunities

- Introspection
- Reduced measurement duration
- Stochastic "extrapolation" of tail latencies

Use Case: cyclictest



System Setup

System

- Kernel v6.11-rc5-rt5
- StarFive VisionFive 2
  - StarFive JH7110 64bit SoC with RV64GC (up to 1.5GHz)
  - LPDDR4 8 GB
  - 4 cores, 1-3 isolated

Steps

- Board Enabling
- RT Tuning
- Implement POC of "Temporal Tracepoints"
- Patch Kernel and cyclictest
- Run measurement
- Fit model
- Derive Predictions

System Setup

System

- Kernel v6.11-rc5-rt5
- StarFive VisionFive 2
  - StarFive JH7110 64bit SoC with RV64GC (up to 1.5GHz)
  - LPDDR4 8 GB
  - 4 cores, 1-3 isolated

Steps

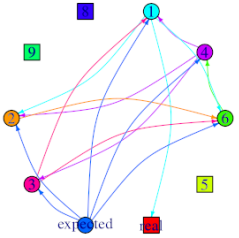
- Board Enabling
- RT Tuning
- Implement POC of "Temporal Tracepoints"
- Patch Kernel and cyclictest
- Run measurement
- Fit model
- Derive Predictions

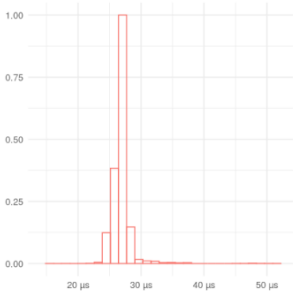
```
        riscv_clock_event_stop();
+       ttp_emit(2);
```

cyclictest: Model

cyclictest

Transition	Latency
1→2	120
2→3	200
3→1	60
1→2	80
2→5	100
5→4	130
4→6	50
6→1	10
1→2	120





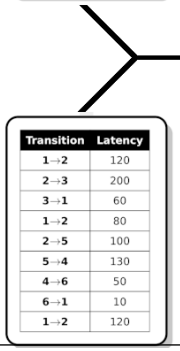
Kernel

Transition	Latency
1→2	120
2→3	200
3→1	60
1→2	80
2→5	100
5→4	130
4→6	50
6→1	10
1→2	120

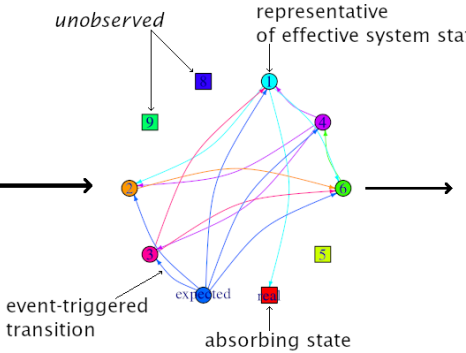
cyclictest: Model

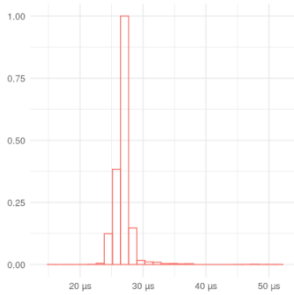
cyclictest

Transition	Latency
1→2	120
2→3	200
3→1	60
1→2	80
2→5	100
5→4	130
4→6	50
6→1	10
1→2	120

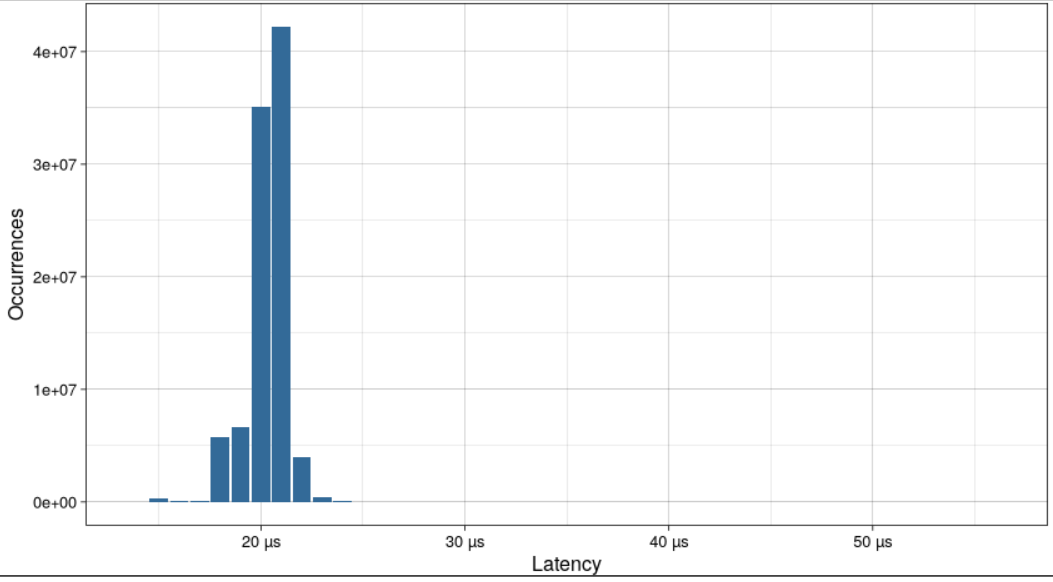


Kernel

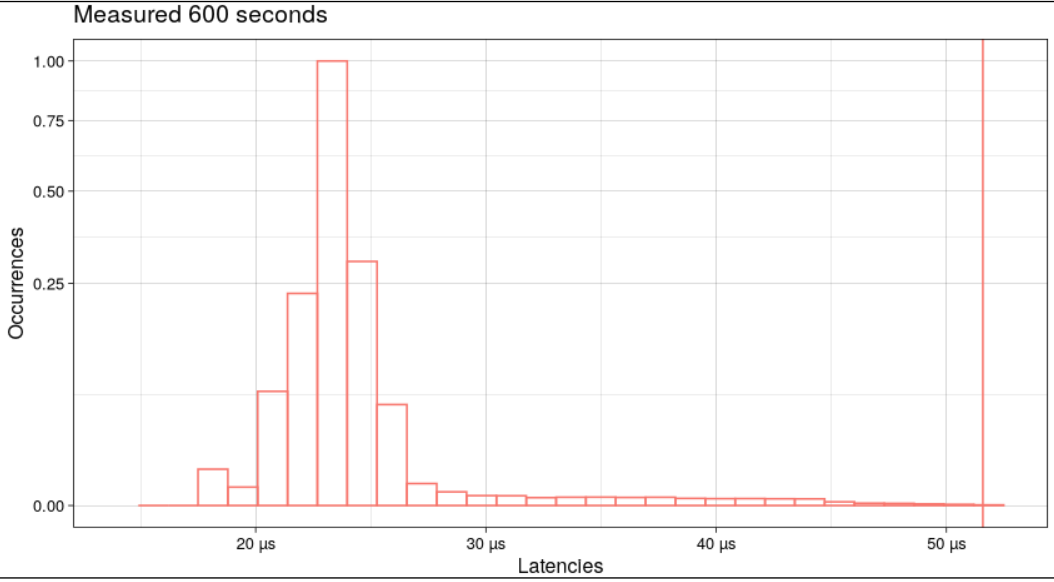




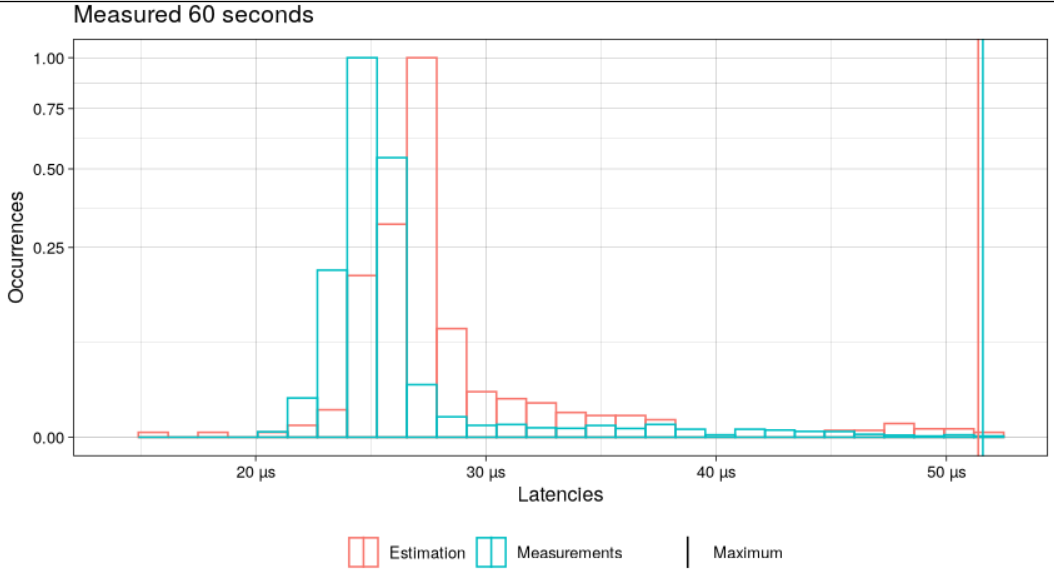
cyclictest – Histogram



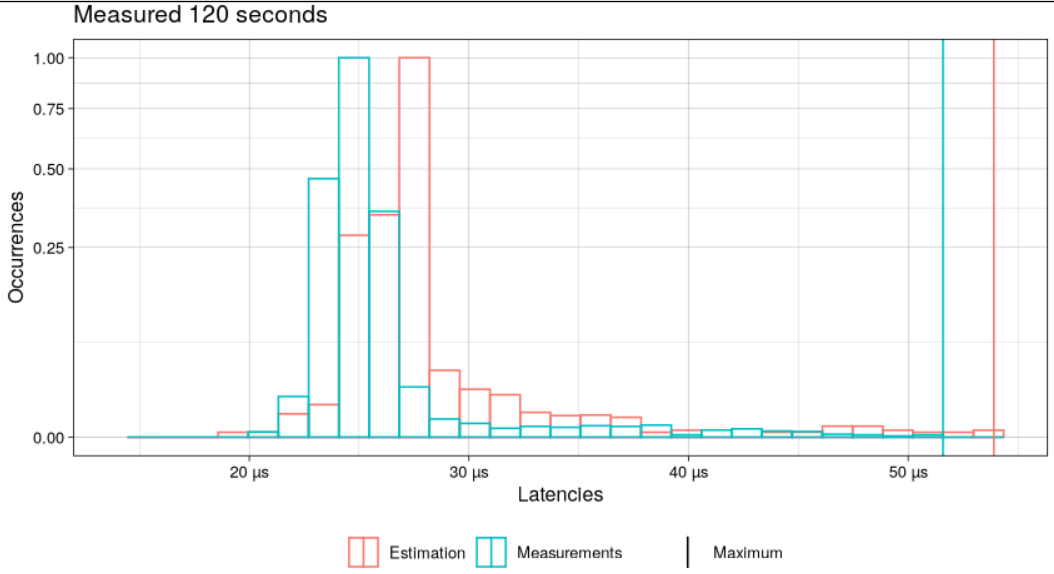
cyclictest – Histogram



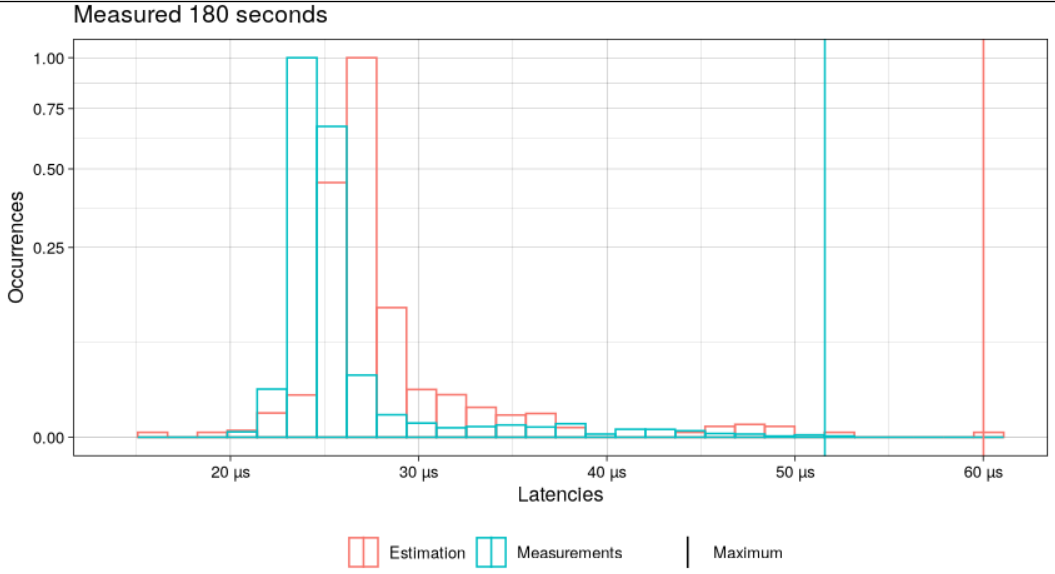
cyclictest – Histogram



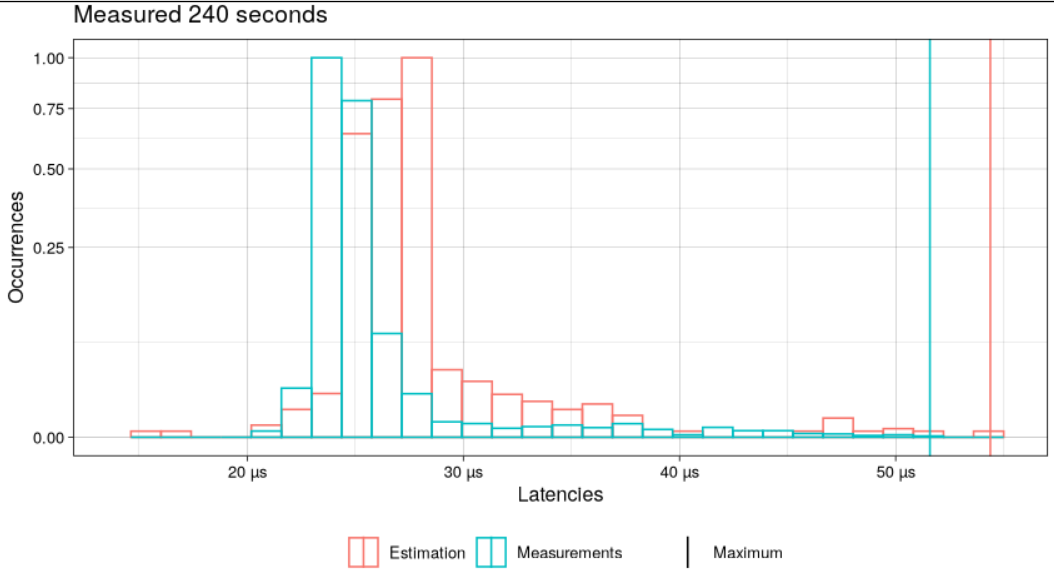
cyclictest – Histogram



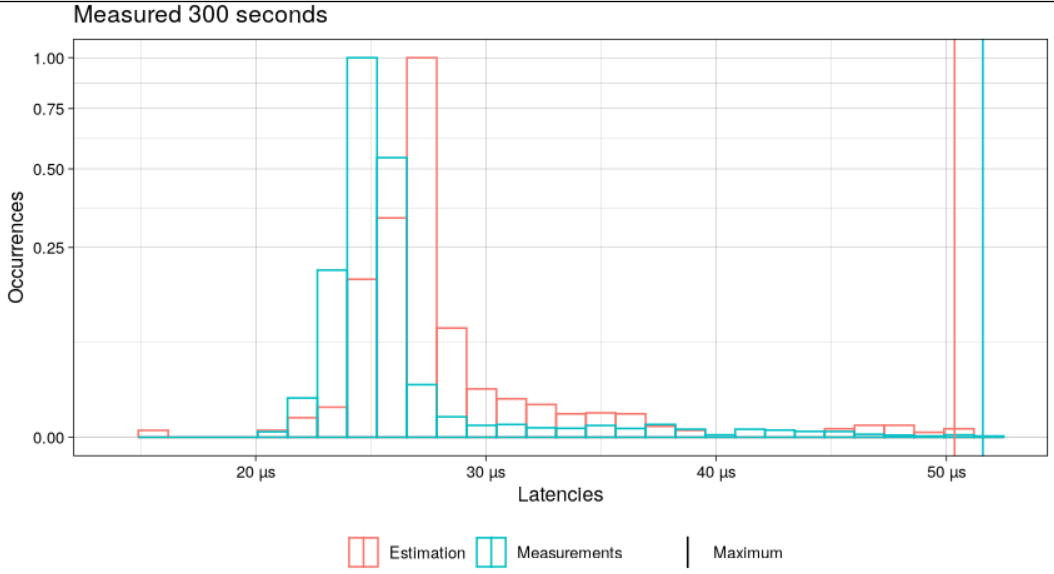
cyclictest – Histogram



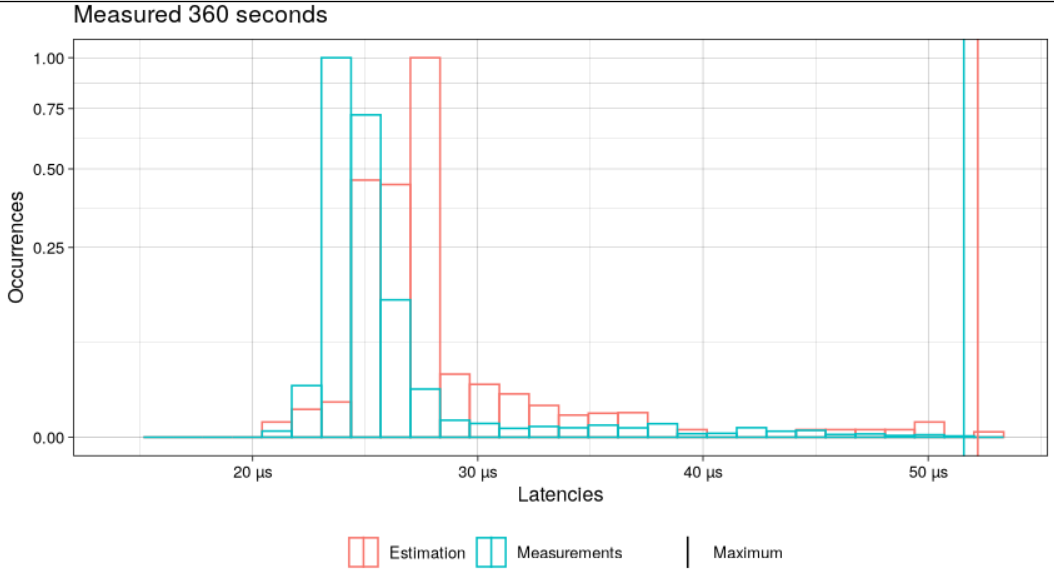
cyclictest – Histogram



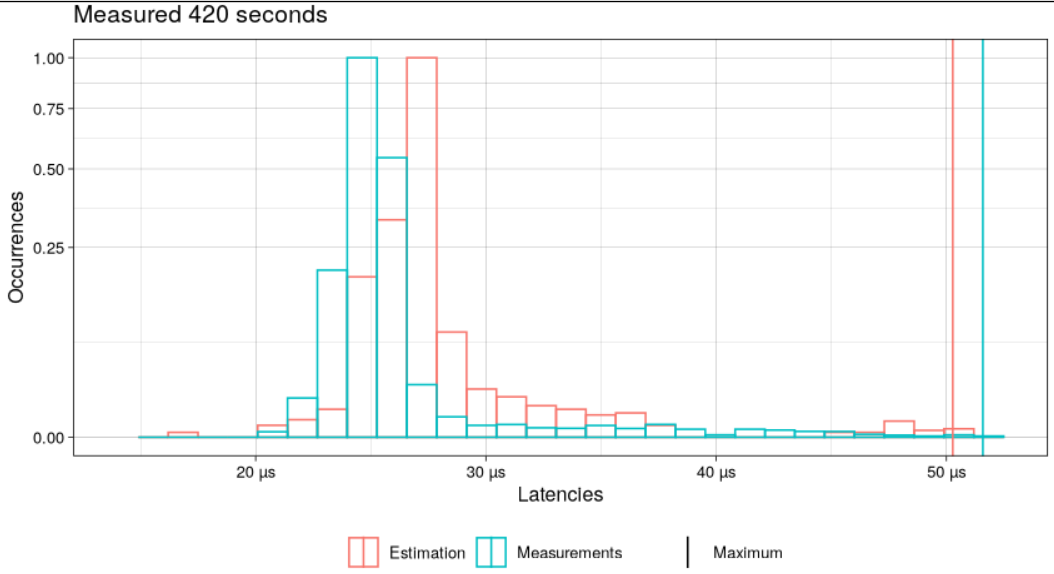
cyclictest – Histogram



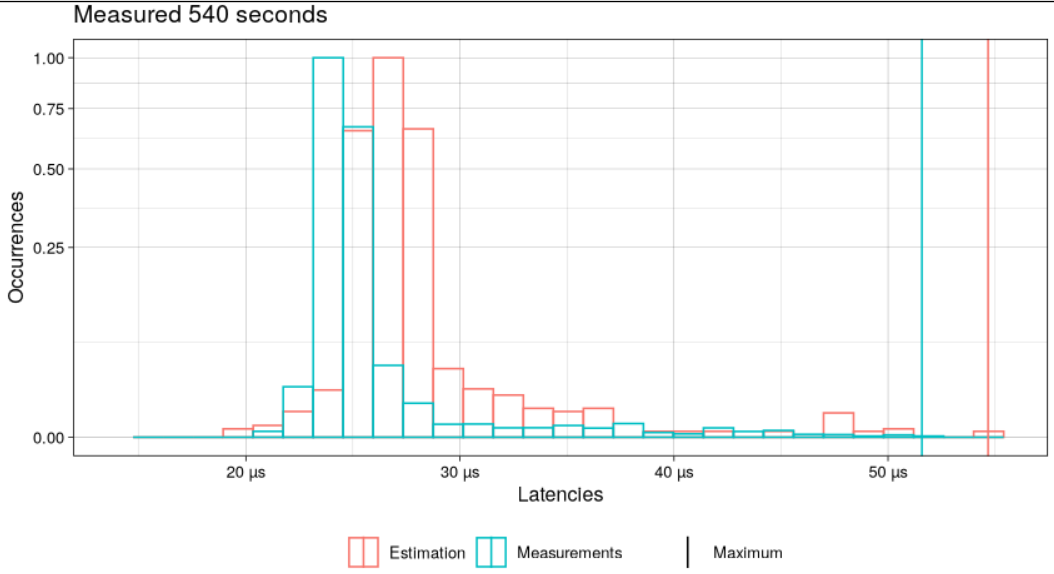
cyclictest – Histogram



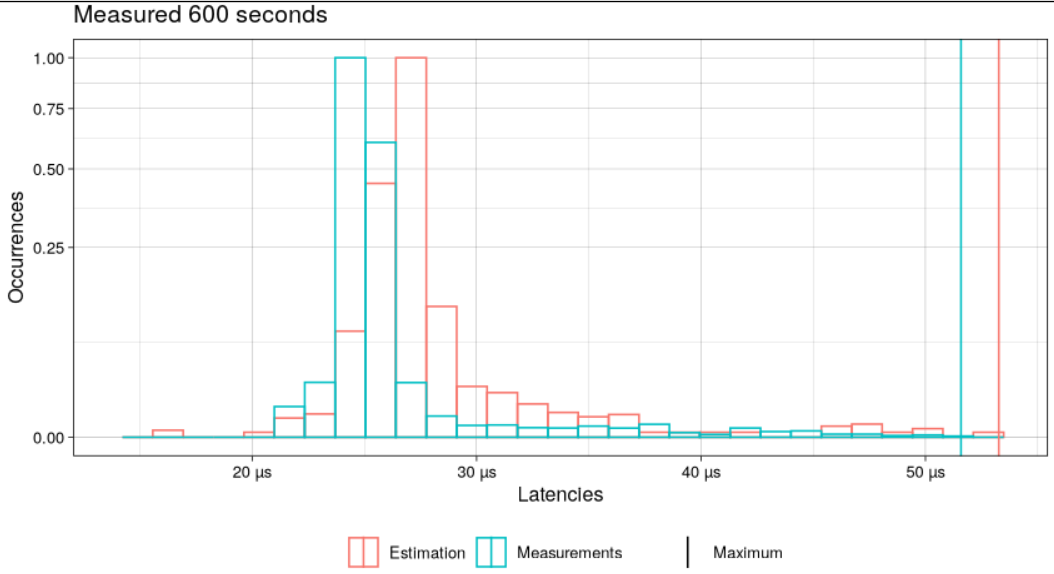
cyclictest – Histogram



cyclictest – Histogram



cyclictest – Histogram



Discussion with the RT folks

Discussion with the RT folks

- Current focus: prediction of WCET. Other applications?

Discussion with the RT folks

- Current focus: prediction of WCET. Other applications?
- Do you incorporate using statistical method?

Discussion with the RT folks

- Current focus: prediction of WCET. Other applications?
- Do you incorporate using statistical method?
- What have been pain points using probabilistic tools?

Thank You!