Automatic Latency Management for ROS 2: Benefits, Challenges, and Open Problems

T. Blass, A. Hamann, R. Lange, D. Ziegenbein, B. Brandenburg
Our Goal: Apply real-time theory to robots

Rapid Development

ROS

Large open-source robotics framework & ecosystem

Timing Correctness

Real-Time Systems Theory

Abstracts from technical details

Mismatch

Needs detailed system model to derive bounds
This Paper in a Nutshell

Why real-time theory is **difficult to apply** to ROS

**Solution:** The live latency manager **ROS-Llama**
- Collect information through **runtime introspection**
- **Automatically** configures real-time scheduler
- **Simple**, declarative specification

**Evaluation:** Case study on a TurtleBot 3
Outline

1. Background
2. Why is real-time theory difficult to apply to ROS?
3. The ROS Live Latency Manager (ROS-Llama)
4. Evaluation
Background: ROS Pub-/Sub Mechanism

Callbacks ➔ Publishes message ➔ Topic /position ➔ Triggers on message arrival ➔ Subscriber callback
Background: Callbacks Form a Graph

Laser Scanner

Topic /scan

Topic /position

Topic /engine_command
Background: ROS Packages

- Topic /scan
- Topic /position
- Topic /engine_command

Laser Scanner
Background: ROS Packages

Developer does not need to know how the package works.
Background: Callback Execution

Developer does not need to know how callbacks are run
Outline

1. Background

2. Why is real-time theory difficult to apply to ROS?

3. The ROS Live Latency Manager (ROS-Llama)

4. Evaluation
Why is real-time theory difficult to apply to ROS?

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Consequence</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ease of use</strong></td>
<td></td>
</tr>
<tr>
<td>• No need to know how included packages work</td>
<td>Cannot ask the user for the information required to bound response times</td>
</tr>
<tr>
<td>• No need to know how callbacks are run</td>
<td></td>
</tr>
<tr>
<td>• No need to know real-time theory</td>
<td></td>
</tr>
<tr>
<td><strong>Below-worst-case provisioning</strong></td>
<td>Must degrade gracefully in case of an overload</td>
</tr>
<tr>
<td><strong>Runs on the officially supported platforms</strong></td>
<td>Must use mainline Linux (+ PREEMPT_RT)</td>
</tr>
</tbody>
</table>
Outline

1. Background

2. Why is real-time theory difficult to apply to ROS?

3. The ROS Live Latency Manager (ROS-Llama)

4. Evaluation
ROS-Llama: Design

Diagram:
- ROS System
- Linux Scheduler
- ROS-Llama
- Declarative Specification
- Trace Events
- Scheduler Configuration
ROS-Llama: Design

- ROS System
- Linux Scheduler
- Trace Events
- Model Extractor
- Scheduler Configuration
- Budget Manager
- Timing Model
- Timing Analysis

Declarative Specification

ROS-Llama
ROS-Llama: Design

ROS System

Trace Events

Model Extractor

Declarative Specification

Scheduler

Configuration

Budget Manager

Timing Model

Linux Scheduler

Timing Analysis

ROS-Llama

Declarative Specification

Timing Model

Budget Manager

Timing Analysis
ROS-Llama: Design

ROS System → Model Extractor → Timing Model → Budget Manager → Timing Analysis

- ROS-Llama
- Trace Events
- Declarative Specification
- Scheduler Configuration
- Linux Scheduler
ROS-Llama: Design

- ROS System
- Trace Events
- Model Extractor
- Timing Model
- Budget Manager
- Timing Analysis

Declarative Specification

- ROS-Llama
- Linux Scheduler
- Scheduler Configuration
ROS-Llama: Design

- ROS System
- Model Extractor
- Trace Events
- Declarative Specification
- Scheduler
- Configuration
- Budget Manager
- Timing Model
- Timing Analysis
- Linux Scheduler
ROS-Llama: Design

- ROS System
- Model Extractor
- Timing Model
- Declarative Specification
- Budget Manager
- Timing Analysis
- Trace Events
- Scheduler
- Configuration

Linux Scheduler
ROS-Llama: Specification

1. End-to-end latency goals for timing-critical processing chains

   - ≤ 60 ms
   - ≤ 70 ms

2. Degradation order

   - In case of overload fail blue chain before red chain
ROS-Llama: Design

- **Declarative Specification**
- **Model Extractor**
- **Timing Model**
- **Budget Manager**
- **Timing Analysis**

- **Trace Events**

- **ROS System**
- **Scheduler Configuration**

- **Linux Scheduler**
The Budget Manager

Step 1: Find relevant callbacks
Which callbacks are part of the designated processing chain?

Executor 1

Executor 2

Topics

Laser Scanner

Topic /scan

Topic /position

Topic /engine_command

≤ 70 ms
The Budget Manager

Step 2: Scheduling Parameters
Which scheduling parameters for the executor threads guarantee the desired response time?

- Executor 1
  - Laser Scanner
  - Topic /scan
  - Topic /position

- Executor 2
  - ≤ 70 ms
  - Topic /engine_command
Background: The SCHED_DEADLINE Scheduler

- Linux’s reservation-based scheduler

- Each thread has a **budget** and a **period**

- Threads are **guaranteed** to receive their budget in each period

- **Provides temporal isolation**
The Budget Manager

Step 2: Choose Budgets/Periods
Which scheduling parameters for the executor threads guarantee the desired response time?

Laser Scanner
Executor 1

Topic /scan

Executor 2

Topic /position

≤ 70 ms

Topic /engine_command
How to Choose Budgets?

Example: Executor A

- Need to bound processor demand of callback 3
  - Depends on number of activations per period
    - Depends on response-time of callback 4
      - Depends on budget of executor B
How to Choose Budgets?

Example: Executor B

- Need to bound processor demand of callback 2
  - Depends on number of activations per period
    - Depends on response-time of callback 1
      - Depends on budget of executor A

Complex global optimization problem

Solution: *greedy heuristic*

**Open Question:** Is there a better solution?
Outline

1. Background
2. Why is real-time theory difficult to apply to ROS?
3. The ROS Live Latency Manager (ROS-Llama)
4. Evaluation
Evaluation Setup

- **Turtlebot 3 “Burger”** controlled by a Raspberry Pi 4B
  - Navigation 2 package
  - ROS 2 Object Analytics package (*Tracker*)
- Robot patrols between two points
- Tracker load **increases over time** in three phases
Evaluation Questions

1. Does ROS-Llama **fulfill** the navigation **latency goals**?

2. What if tracker load increases (**graceful degradation**)?

3. Would a **simpler solution** do just as well?

- **CFS** (fair-share scheduler)
  - Default Linux scheduler, no configuration required

- **SCHED_RR** (fixed-priority scheduler)
  - Criticality-monotonic priorities for controlled degradation
Evaluation Results: Pilot Chain

Only ROS-Llama manages latency well in all situations
Evaluation Results: Pilot Chain

Only **ROS-Llama** manages latency well in all situations

**General-purpose** scheduling may work better than misconfigured real-time scheduling
Evaluation Results: Pilot Chain

Only ROS-Llama manages latency well in all situations

General-purpose scheduling may work better than misconfigured real-time scheduling

Impact of load is complex and hard to predict without analysis
Summary

**ROS-Llama**: Easy, low-effort real-time scheduling for ROS

- Simple declarative specification
- Automatic scheduler configuration at runtime
- In the paper: extensive discussion of open problems
  - SCHED_DEADLINE limitations
  - Linux I/O
  - ROS Idiosyncrasies
  - Complex Activations
  - Stochastic Analysis
  - DDS Analysis
Applying real-time theory in robotics is difficult

Rapid Development

Large open-source robotics framework & ecosystem

Timing Correctness

Real-Time Systems Theory

Well-established theory

Abstracts details for simpler understanding and reuse

Use detailed description of the system to derive strong bounds