Response-Time Analysis of ROS 2 Processing Chains under Reservation-based Scheduling

Daniel Casini, Tobias Blass, Ingo Lütkebohle, Björn Brandenburg
This Paper in a Nutshell

Robots are **complex cyber-physical systems**, subject to real-time constraints

**ROS**, the most popular robotics framework is **not based on real-time system models** and **no response-time analysis exists**

**This work:**
Understand the surprising and undocumented timing behavior of ROS
Develop a response-time analysis that is aware of ROS’s quirks

https://robots.ros.org/robonaut2/

https://robots.ros.org/innok-heros/
The Robot Operating System (2007)

Popular robotics **framework** in academia and industry

- > 10 000 users
- > 1000 packages
- > 500 attendees at ROSCon

12 years of development exposed many **limitations** in the original design

*Willow Garage PR2, the original ROS robot*

http://rasc.usc.edu/pr2.html
From ROS to ROS 2

- Complete refactoring of the ROS framework
- Recently released first long-term support version
- Aims to support **real-time control**

“We want to support real-time control directly in ROS, including inter-process and inter-machine communication”

From “Why ROS 2?” at design.ros2.org

What does this mean in practice?

https://www.youtube.com/watch?v=npQMzH3j_d8
Our Needs for Real-Time Control

1. Support for automated timing **validation**
   - Does this robot react in time?

2. Support for model-based **design-space exploration**
   - Would this robot react in time if I used this hardware?
   - Would this robot react in time if I implemented it that way?

We need a response-time analysis
The Quest for a ROS 2 Response-Time Analysis
ROS Systems are distributed networks of callbacks
The Quest for a ROS 2 Response-Time Analysis

Callbacks are assigned to *executor threads*.

**Callback timer** triggers

**ROS-Level Scheduling**  
- Callback

**Linux-Level Scheduling**
- Thread 1
- Thread 2
Is callback 2 or callback 4 executed first?

The ROS documentation does not specify the execution order of callbacks.
The Quest for a ROS 2 Response-Time Analysis

Looking at the source code

The four layers of the ROS implementation

<table>
<thead>
<tr>
<th>Language-specific Client Libraries</th>
<th>rclcpp</th>
<th>rclpy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Library</td>
<td>rcl</td>
<td></td>
</tr>
<tr>
<td>Middleware Library</td>
<td>rmw</td>
<td></td>
</tr>
<tr>
<td>DDS Implementations</td>
<td>RTI Connext DDS</td>
<td>OpenSplice DDS</td>
</tr>
<tr>
<td></td>
<td>eProsima FastRTSP</td>
<td></td>
</tr>
</tbody>
</table>

**Question**: Which layer determines the callback execution order?

**Answer**: All of them
The Quest for a ROS 2 Response-Time Analysis
Looking at the source code

The four layers of the ROS implementation

<table>
<thead>
<tr>
<th>Language-specific Client Libraries</th>
<th>rclcpp</th>
<th>rclpy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Library</td>
<td>rcl</td>
<td></td>
</tr>
<tr>
<td>Middleware Library</td>
<td>rmw</td>
<td></td>
</tr>
<tr>
<td>DDS Implementations</td>
<td>RTI Connext DDS</td>
<td>OpenSplice DDS</td>
</tr>
<tr>
<td></td>
<td>eProsima FastRTPS</td>
<td></td>
</tr>
</tbody>
</table>

Callback execution order is **ROS-specific** and combines properties from:
- Fixed-priority scheduling
- FIFO scheduling
- TDMA scheduling
Why not use a framework that prioritizes real-time?

<table>
<thead>
<tr>
<th></th>
<th>ROS</th>
<th>real-time robotics frameworks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Community size</td>
<td>Huge</td>
<td>Small</td>
</tr>
<tr>
<td>Effort to integrate third-party software</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Simulation support</td>
<td>Out-of-the-box, with ready-made models for many parts</td>
<td>None</td>
</tr>
<tr>
<td>Hardware Support</td>
<td>Lots of robotics hardware comes with ROS drivers</td>
<td>low-level Linux drivers</td>
</tr>
<tr>
<td>Predictability</td>
<td>Difficult</td>
<td>Easy</td>
</tr>
</tbody>
</table>

Guess which one people use?
This Work
Make ROS 2 more predictable by

Understanding and documenting the ROS 2 timing behavior

Developing an end-to-end response-time analysis for ROS 2
This Work
Make ROS 2 more predictable by

Understanding and documenting the ROS 2 timing behavior

Developing an end-to-end response-time analysis for ROS 2
ROS Systems

ROS Systems consist of callbacks, grouped into nodes.
ROS Systems

Nodes communicate using topics (a pub/sub mechanism) ...

Nodes communicate using topics (a pub/sub mechanism) ...
ROS Systems

... and services (remote procedure calls)
ROS Systems

Nodes are assigned to executor threads
ROS Systems

The operating system’s view
Understanding ROS’s Timing Behavior

Who determines the execution order?

- Linux Scheduler
- Controls executor scheduling
- Controls callback order
- Node
- Node

OS Process

- IPC Layer
- Topics
- Services
- Timers
- Executor Thread
- Executor Thread

Executor

© Robert Bosch GmbH 2018. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Understanding ROS’s Timing Behavior
Who determines the execution order?

Linux Scheduler

ROS-specific, undocumented

Well understood (SCHED_DEADLINE)
The Executor’s Algorithm

Polling Point

readySet := ready sources in IPC Layer

Processing Window

Execute highest-priority callback
Remove callback from readySet

timer ready?

yes

no

topic in readySet?

yes

no

service in readySet?

yes

no

service reply in readySet?

yes

no

yes

no
Peculiarities of the Executor

readySet := ready sources in IPC Layer

Events only noticed at the next polling point

Execute highest-priority callback
Remove callback from readySet

timer ready?

no

except for timers

yes

topic in readySet?

yes

no

service in readySet?

yes

no

service reply in readySet?

yes

no
Peculiarities of the Executor

readySet := ready sources in IPC Layer

- Window length is dynamic ⇒ Cannot predict polling points
- Execute highest-priority callback
- Remove callback from readySet

timer ready?
- yes
- no

- topic in readySet?
  - yes
  - no

- service in readySet?
  - yes
  - no

- service reply in readySet?
  - yes
  - no
Peculiarities of the Executor

Only one event per processing window per source

readySet := ready sources in IPC Layer

Execute highest-priority callback
Remove callback from readySet

except for timers

timer ready?

no

readySet := ready sources in IPC Layer

topic in readySet?

no

service in readySet?

no

service reply in readySet?

yes
The Executor: Summary

The ROS 2 Executor **differs significantly from the usual schedulers**

Existing models and analyses cannot be applied directly:

- Existing models don’t capture the necessary information
- Existing analyses don’t consider the quirks of the ROS 2 Executor

The artifact contains an experiment that validates our model of the executor’s behavior
This Work
Make ROS 2 more predictable by

Understanding and documenting the ROS 2 timing behavior

Developing an end-to-end response-time analysis for ROS 2
Response-Time Analysis
Compositional Performance Analysis (CPA)  
Henia et. al., 2005

Per-Task Analysis
Computes per-task response times
given event arrival curves

Arrival-Curve Propagation
Computes event arrival curves
given per-task response times

Fixed-point search

CPU

task

CPU

task

CPU

task

CPU

task
Response-Time Analysis
The CPA approach fits ROS well

CPU

task

CPU

task

Executor reservation

callback

callback

Executor reservation

callback

callback
Response-Time Analysis
Extending CPA for ROS

Per-Callback Analysis

Needs to account for ROS’s quirks

Fixed-point search

Arrival-Curve Propagation

Inherited from CPA unchanged

Executor

callback

Executor

callback

callback

Needs to account for ROS’s quirks

Fixed-point search

Inherited from CPA unchanged
Response-Time Analysis
A real-time model for ROS 2

**Timer callback**
- WCET bound $e_i$
- Priority $\pi_i$
- Event arrival curve $\eta_i^g$

**Subscription callback**
- WCET bound $e_i$
- Priority $\pi_i$

**Processing chains of interest**

**Communication Delay** $\delta_{i,j}$

**Event source external to ROS**
- Event arrival curve $\eta_i^a$
Response-Time Analysis
A per-callback response-time analysis for ROS

\[ sbf_k(A + R_i^r(A)) = rbf_i(A + 1) + RBF(hp_k(c_i), A + R_i^r(A) - e_i + 1) + B_i \]

\[ sbf_k(A + R_i^r(A)) = rbf_i(A + 1) + RBF(\{C_k \setminus c_i\}, A + R_i^r(A) - e_i + 1) \]

Dedicated analyses for **timers and polling-point-based callbacks**

Busy-window analysis **without critical instant assumptions**

Optimization for **intra-executor** chains
Case Study: The *move_base* Navigation Stack
Processing Steps

Can we implement this safely in ROS 2?
Our Needs for Real-Time Control

1. Support for automated timing validation
   - Does this robot react in time?

2. Support for model-based design space exploration
   - Would this robot respond in time if I used this hardware?
   - Would this robot respond in time if I implemented it that way?
Hardware Uncertainty

- odometry
- position
- laser scanner

 pose → Local map → Local planner → wheels

⇒ Unknown input jitter

Hardware not determined yet
Two Implementation Choices

**Time-Driven**
- odometry
- position
- laser scanner
- store
- pose
- Local map
- Local planner

**Event-Driven**
- odometry
- position
- laser scanner
- store
- pose
- Local map
- Local planner
- Global map
- Global planner

How well do both variants cope with increasing input jitter?
Exploring Jitter Sensitivity

![Graph showing End-to-End latency bound (ms) against Jitter on the input sensors (ms). The graph compares time-driven and event-driven approaches. The event-driven approach shows a significant increase in latency with increasing jitter, while the time-driven approach remains relatively stable.]
Exploring Jitter Sensitivity

fixed sampling latency

Not affected by jitter
Exploring Jitter Sensitivity

Increasing self-interference

no sampling latency
Exploring Jitter Sensitivity

“All laser scanners have jitter <40ms”
⇒ event-driven approach yields faster response
Exploring Jitter Sensitivity

Response-time analysis allows **nonobvious tradeoffs** **early** in the design phase.
Future Work

Extract the timing model automatically
  static analysis or runtime system introspection

Develop a real-time executor
  Discussions with ROS developers underway (Join us!)
Summary

Contributions

- A comprehensive description of the ROS 2 execution model
- A response-time analysis for ROS 2 applications

This work enables ROS users to

- Explore the design space cheaply and early
- Provide safety guarantees for their applications

This work provides real-time researchers with

- A task model that reflects the leading robotics framework

Source code available at https://github.com/boschresearch/ros2_response_time_analysis