Message Flow Analysis for Distributed ROS 2 Systems

Christophe Bourque Bédard

Progress Report Meeting
May 16, 2022

Polytechnique Montréal
DORSAL Laboratory

POLYTECHNIQUE MONTRÉAL
TECHNOLOGICAL UNIVERSITY

Summary

1. Introduction
2. ROS 2
3. ROS 2 executor & scheduling
4. Message flow analysis
5. Experiments
6. Runtime overhead evaluation
7. Conclusion and future work
8. Questions

Introduction

- Robotics
  - Commercial or industrial applications
  - Safety-critical applications
  - Can be **distributed** and connected over a network (e.g., 5G)
- Key elements
  - Message passing (publish-subscribe) and Remote Procedure Call (RPC)
  - Performance targets, real-time constraints
  - **Higher-level scheduling of tasks is challenging**
- Robotics software development can greatly benefit from tracing

ROS 2

- Robot Operating System 2
  - docs.ros.org/en/humble
- Open source framework and set of tools for robotics software development
  - Well-known in robotics
  - Used as *Space ROS* for NASA's 2023 Moon rover, VIPER 🚀🌙
- Message passing between “nodes”
  - Publish/subscribe
  - Service/action calls (~RPCs)
- Modular
  - Each node generally accomplishes a very specific task
  - Nodes are put together to perform complex tasks
- Uses Data Distribution Service (DDS) as the middleware
  - OMG standard
- Intra-process, inter-process, and distributed

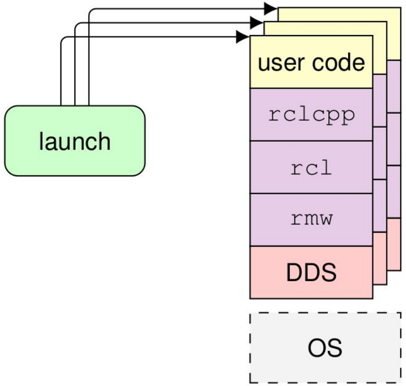

Figure 1. ROS 2 architecture and orchestration.

ROS 2 executor & scheduling

- Executor
  - High-level task scheduler
  - Fetches new messages from underlying middleware
  - Executes user-provided timer and subscription callbacks
- Challenges
  - Prioritizes timers first, then subscriptions
  - Scheduling on top of the OS scheduler can be inefficient & non-deterministic
- Possible solutions
  - Other executor designs, depending on the application/requirements
  - Optimize scheduling policies and priorities
- Need to study and compare executors
  - And optimize overall application performance

Trace data processing

- Distributed systems
  - Combine traces
  - Synchronize traces using NTP, PTP, or offline sync using Trace Compass
- Modeling ROS 2 objects and instances from trace data
  - Using pointers as unique IDs
  - Combine with PID and host ID
- Model
  - Objects: nodes, publishers, subscriptions, timers, etc.
  - Instances: message publications, timer & subscription callbacks, etc.
- Can use this pre-processed data to extract further metrics or provide other views

Message flow analysis

- Graph of the path of a message across a distributed ROS 2 system
  - Combine multiple segments and links
- Subscription and timer callbacks
- Message publication instances
- Transport links
- **Causal message links**: primarily based on message data

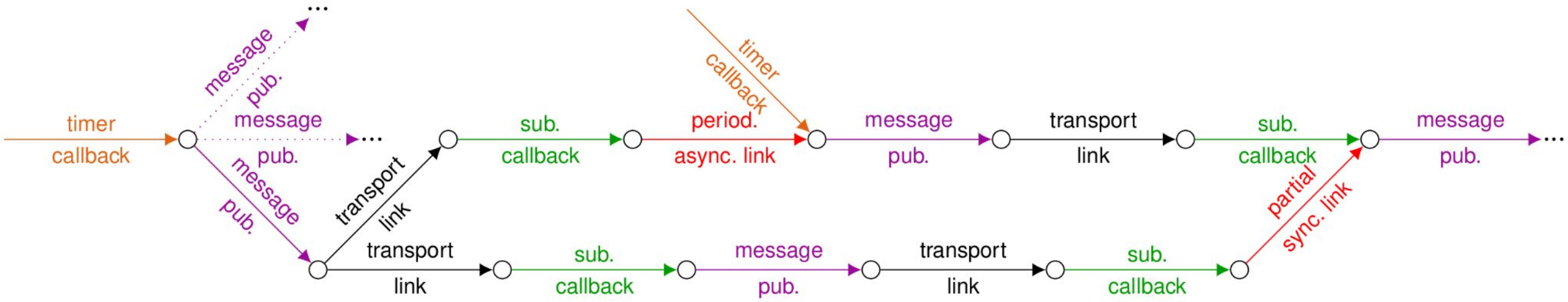

Figure 2. Simplified representation of a message flow graph.

Message flow analysis (2)

- Transport link
  - Link between publication instance and corresponding subscription callback
- Includes more than just network time
  - Delay between message reception and callback execution
- One-to-many link
  - 1 publisher → N subscriptions

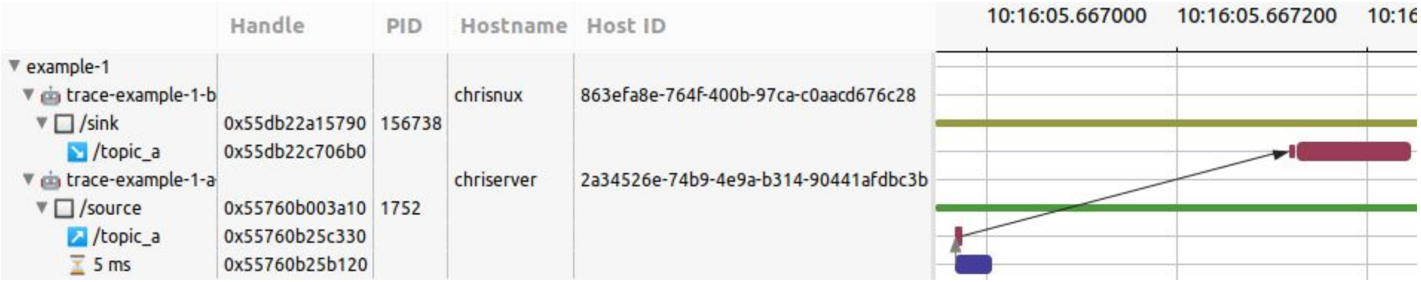

Figure 3. Transport link from one host to another host.

Message flow analysis (3)

- Direct causal link
  - Message publication during subscription callback for message
- Can be inferred automatically
  - No need for additional information or instrumentation

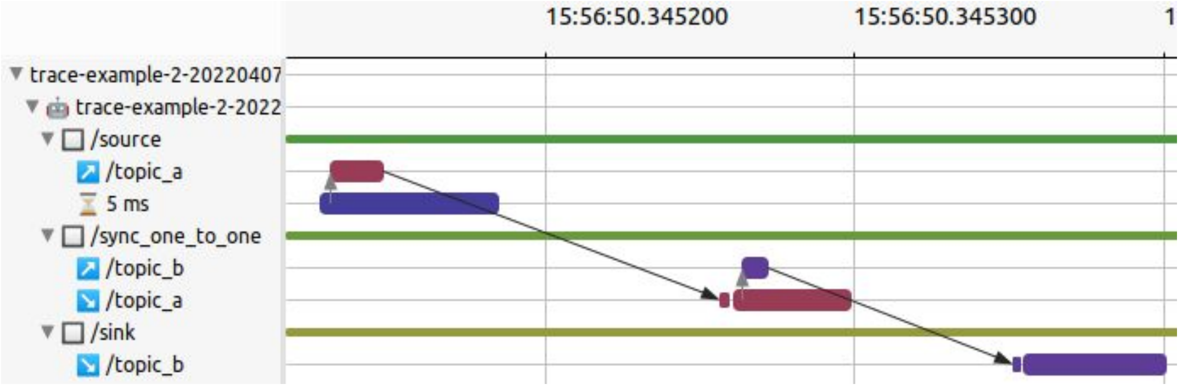

Figure 4. Direct causal link.

Message flow analysis (4)

- Indirect causal link: **asynchronous**
- Requires additional user-level annotation
  - Collected using simple tracepoints

Figure 5. Indirect causal link: timer callback uses last received messages.

Message flow analysis (5)

- Message flow graph
- Can extract
  - End-to-end latency
  - Intermediate latencies
- Can visually understand execution
  - Find bottlenecks

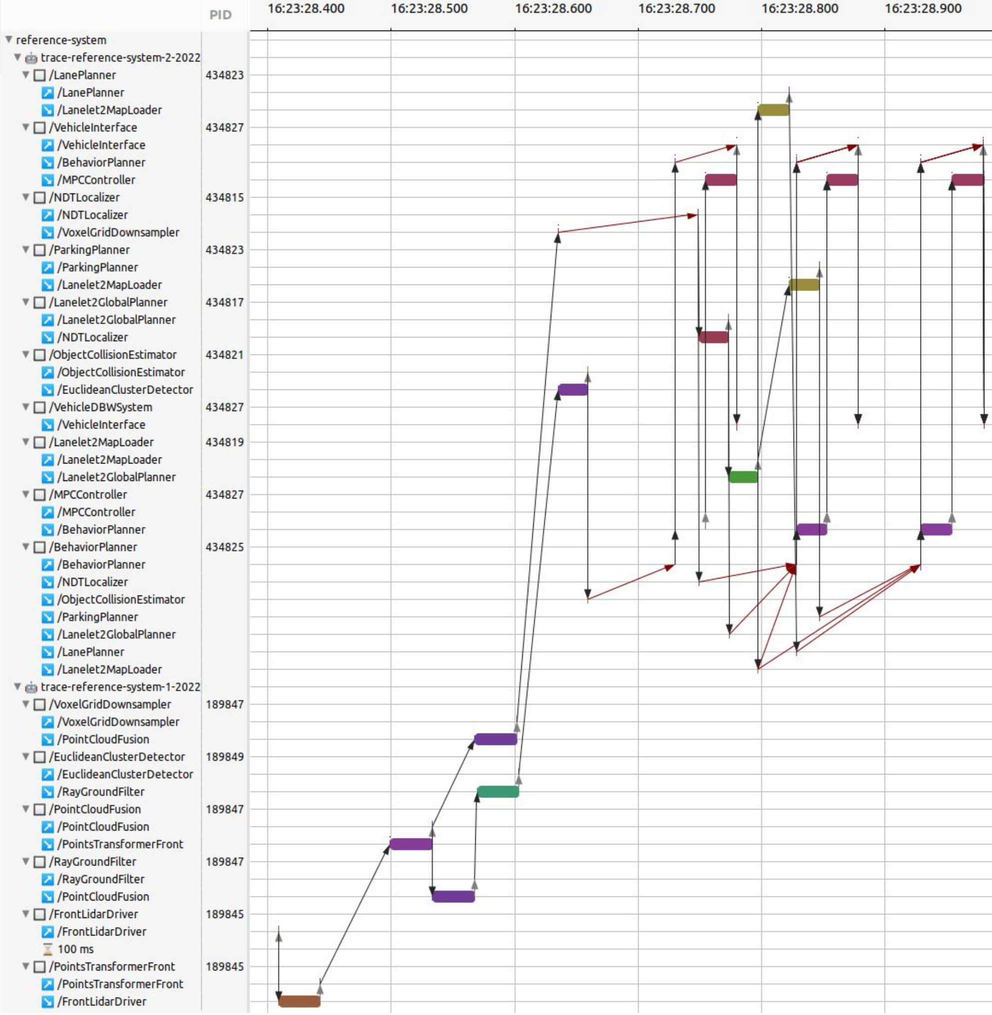

Figure 6. Message flow analysis result example.

Executor state

- Green/orange: executing/waiting for new messages or timer trigger
- Some executor instances are busier than others
- Causes message processing delays, leads to bottlenecks
- Possible solutions: multi-threaded executor, thread priorities

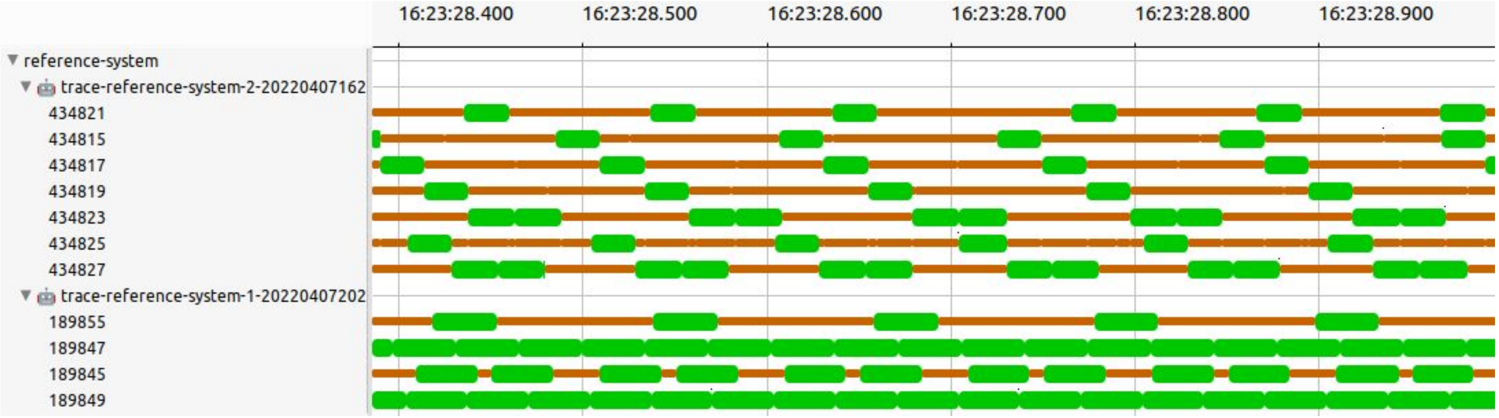

Figure 7. View showing state of executor instances (threads) over time.

Runtime overhead evaluation

- Extrapolating from previous overhead results
  - Should be very small
- Execute pipeline of nodes, without & with tracing
  - Total end-to-end latency of ~260 ms
- Overhead is the difference
  - Difference of means : 0.1597 ms
  - Difference of medians: 0.0521 ms
- Likely challenging to measure on more complex systems

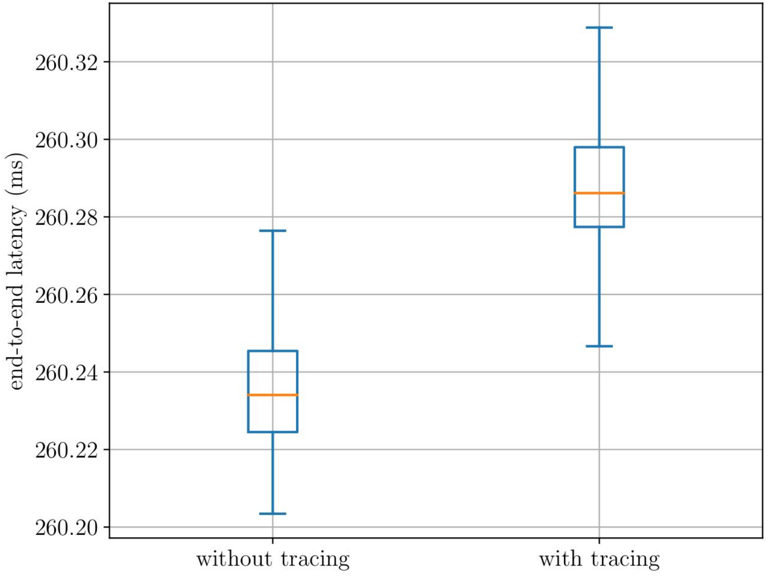

Figure 8. End-to-end latency comparison.

Conclusion and future work

- Tracking messages across nodes
  - Building a message flow graph using this information
  - Using user-level annotation to find more complex indirect causal links
- Computing end-to-end latency
- Study and improve performance of an application and ROS 2 itself

- Future work
  - Resolve wait dependencies resulting from asynchronous causal links
  - Critical path analysis at the ROS 2 level
  - Augment graph with other information: application-level or kernel-level

Questions?

- christophe.bedard@polymtl.ca
- Links
  - docs.ros.org/en/humble
  - gitlab.com/ros-tracing/ros2_tracing
  - ROS 2 message flow paper (in review)
    - Message Flow Analysis with Complex Causal Links for Distributed ROS 2 Systems
    - arxiv.org/abs/2204.10208
  - ros2_tracing paper in *IEEE Robotics and Automation Letters*
    - ros2_tracing: Multipurpose Low-Overhead Framework for Real-Time Tracing of ROS 2
    - ieeexplore.ieee.org/document/9772997
    - arxiv.org/abs/2201.00393
- Other relevant links
  - Presentation at a ROS conference in 2021
    - vimeo.com/652633418 (slides)

Tracing ROS 2

- Tools part of the ROS 2 core
  - gitlab.com/ros-tracing/ros2_tracing
- LTTng instrumentation in ROS 2
  - Message publication & reception
  - Subscription & timer callbacks
  - Etc.
  - Constant number of trace events, constant overhead (?)
- And some LTTng instrumentation for a DDS implementation
- Tracing tools closely integrated with ROS 2
  - ROS 2 CLI tools
  - ROS 2 launch/orchestration system

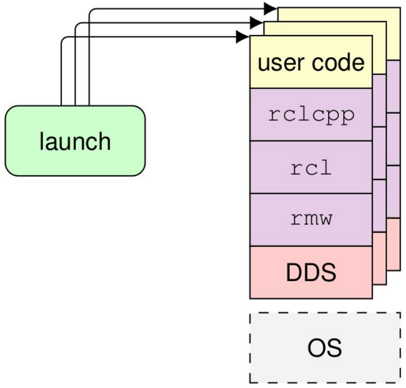

Figure 9. ROS 2 architecture and orchestration.