End-To-End Timing Analysis and Optimization of Multi-Executor ROS 2 Systems

Harun Teper*, Tobias Betz†, Mario Günzel*, Dominic Ebner†,
Georg von der Brüggen*, Johannes Betz† and Jian-Jia Chen*‡
**TU Dortmund University*, Germany, †*Technical University of Munich*, Germany , ‡*Lamarr Institute*, Germany
{harun.teper, mario.guenzel, georg.von-der-brueggen, jian-jia.chen}@tu-dortmund.de
{tobi.betz, dominic.ebner, johannes.betz}@tum.de

***Abstract*—Modern robot systems, like autonomous vehicles, are complex, distributed systems that consist of many interacting components. End-to-end timing latency guarantees are key properties of such systems. They upper bound the data processing time and provide a predictable timing behavior. The Robot Operating System 2 (ROS 2) is a widely used and highly configurable set of software libraries for creating and deploying robot systems. It features a custom scheduler to execute time-triggered and event-triggered tasks and uses Data Distribution Services (DDS) for the communication between different system components. The data propagations between ROS 2 system components form cause-effect chains, which can be analyzed to determine the maximum reaction time (longest time between occurrence of an external cause and the earliest time when this external cause is fully processed) and maximum data age (longest time between the moment of a sensor measurement and the latest moment where an effect is based on this sensor measurement).**

In this paper, we provide an analysis of the end-to-end latencies in multi-executor ROS 2 systems to upper bound the end-to-end latencies of cause-effect chains in ROS 2 systems. Furthermore, we introduce an optimization using constrained programming that determines the optimal system configuration to minimize the end-to-end latencies for ROS 2 systems. We evaluate our upper-bound analysis to determine the end-to-end latencies of cause-effect chains in an autonomous driving-software stack for oval racing used in the Indy Autonomous Challenge and apply our optimization method to reduce the end-to-end latency upper bound, measured maximum, and measured mean by up to $50.2\,\%$, $19.8\,\%$, and $7.2\,\%$, respectively.

***Index Terms*—End-to-End Timing, Maximum Reaction Time, Maximum Data Age, Robot Operating System 2**

I. Introduction

Modern robot systems consist of many interacting components that process data from sensors and output it to actuators. These, often safety-critical, systems require end-to-end timing guarantees to ensure safe and predictable behavior.

The Robot Operating System 2 (ROS 2) [20] is a widely used middleware that provides software libraries and tools for building and deploying highly configurable robot systems. The components of the system include time-triggered and event-triggered tasks, which are scheduled by a custom scheduler abstraction called *executor*. Furthermore, the system components communicate via Data Distribution Services (DDS) [21], which provide a publish-subscribe architecture as the data-propagation backbone. Depending on the system and DDS configuration, the latency for data propagation between components can vary significantly.

In comparison to the original Robot Operating System (ROS) [22], the ROS 2 executor and DDS communication provide the possibility for real-time guarantees. So far, the focus of real-time analysis in ROS 2 has been on individual tasks, analyzing the worst-case response time (WCRT) using DAG-based methods [7], [8] and improving the WCRT using priority assignments [9], [27]. Additional executor designs and prioritization policies were proposed and analyzed to improve the response time of ROS 2 systems [1], [17], [26], [31].

Recently, the focus shifted to end-to-end timing analysis of cause-effect chains, which considers the data-propagation path from sensor to actuator. Originating from the event-chains of the AUTOSAR Timing Extensions [2], a cause-effect chain describes a sequence of tasks from a cause (an external signal captured by a sensor) to an effect (caused by an actuator output). Teper et al. [29] analyzed end-to-end timing metrics — namely, the maximum reaction time (longest time between an external cause and the earliest time when this external cause is fully processed) and the maximum data age (longest time between a sensor measurement and the latest moment where an effect is based on this sensor measurement) — for cause-effect chains in single-executor ROS 2 systems.

However, their analysis did not consider systems with multiple executors, the de facto standard for ROS 2 systems, and the effects of the system configuration on end-to-end latencies.

Contributions: We provide an end-to-end timing analysis for cause-effect chains in ROS 2 systems with multiple single-threaded executors. Moreover, we introduce an optimization approach to minimize end-to-end latencies of cause-effect chains. This paper provides the following contributions:

- Section VI provides an upper-bound analysis for the maximum reaction time and maximum data age of cause-effect chains in *multi-executor* ROS 2 systems.
- In Section VII, we detail how the end-to-end latencies of cause-effect chains in ROS 2 systems can be optimized using constrained programming.
- We evaluate our analysis and optimization, using a software stack for autonomous racing [4], in Section VIII.

We detail the ROS 2 system and communication model in Section II and introduce ROS 2 scheduling in Section III. The ROS 2 latencies are detailed in Section IV, while Section V introduces end-to-end data propagation in ROS 2 systems.

TABLE I
NOTATIONS IN THIS PAPER.

Notation	Description
$\mathcal{N}$	Set of nodes
$\mathcal{T}$	Set of tasks
$\mathcal{X}$	Set of executors
n_i	Node i
τ_i	Task i
e_i	Executor i
E_i	Cause-effect chain i
C_i^{τ}	WCET of task τ_i
tmr_i	Timer i
T_i	Period of timer tmr_i
sub_i	Subscription i
st_i	Subscription topic of subscription sub_i
K_i	Buffer size of subscription sub_i
cb_i	Callback i
pt_i	Set of publisher topics of callback cb_i
wl_i	Set of write labels of callback cb_i
rl_i	Set of read labels of callback cb_i
ρ	Index for processing windows
P_ρ	Polling point of processing window with index ρ
$J_{i,\rho}$	Job of task τ_i in processing window with index ρ
$s_{i,\rho}$	Start time of job $J_{i,\rho}$
$f_{i,\rho}$	Finish time of job $J_{i,\rho}$
$\pi_{i,j}$	Binary priority comparison between task i over task j
$\delta_{i,t}^{dds}$	DDS thread time for DDS publication to topic t_i for jobs of task τ_i
$\vec{ac}_z$	Immediate forward augmented job chain at time z
$\overleftarrow{ac}_{z'}$	Immediate backward augmented job chain at time z'

II. ROS 2 System and Communication Model

ROS 2 is a middleware for creating robot systems that consist of multiple interacting system components, like planning, localization, and mapping for autonomous navigation. This section provides an overview of ROS 2 and introduces a system model for ROS 2 systems based on ROS 2 Humble [20] (the current LTS stable release of ROS 2) and the eProsima Fast DDS implementation of the DDS standard [12], as it is highly configurable and provides lower communication latencies compared to other available DDS solutions [6].

The ROS 2 system components are specified as a set of nodes $\mathcal{N} = \{n_1, \dots, n_{|\mathcal{N}|}\}$. Each node n_i includes a set of tasks, and each task is uniquely assigned to one node. We denote the complete set of tasks over all nodes as $\mathcal{T} = \{\tau_1, \dots, \tau_{|\mathcal{T}|}\}$. Figure 1 depicts a small example system consisting of two nodes with three tasks each.

A task τ_i in ROS 2 is either *time-triggered*, called a timer tmr_i, or *event-triggered*, called a subscription sub_i. When a task τ_i is scheduled (see Section III for ROS 2 scheduling), a corresponding function, called callback cb_i, is executed.

In ROS 2, tasks can communicate in different ways. We differentiate between the communication of tasks assigned to the same node, called *intra-node communication*, and the communication of tasks assigned to different nodes, called *inter-node communication*. For intra-node communication, tasks may communicate either via node variables, referenced by labels l_i (indicated by black arrows in Figure 1), or via DDS (indicated by colored arrows in Figure 1). Inter-node communication can only occur through DDS.

A Data Distribution Service (DDS) is a standard for publish-subscribe communication. It uses so-called topics for the communication via messages between tasks. Timer callbacks and subscription callbacks can publish messages to topics.

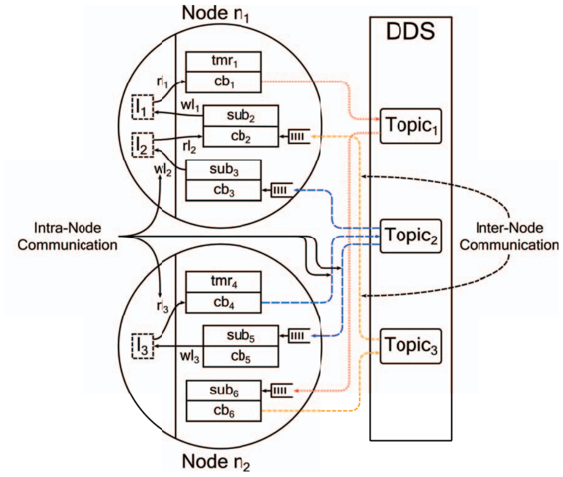

Fig. 1. ROS 2 system components and communication overview.

These messages are received by all subscriptions that are subscribed to that topic. DDS messages are the events that trigger subscription tasks. In contrast, label-based communication does not directly trigger tasks in ROS 2.

In ROS 2, timers and subscriptions have different properties. As shown in Figure 1, each task τ_i includes a callback, defined as a tuple $cb_i = (C_i^{cb}, pt_i, wl_i, rl_i)$. The worst-case execution time (WCET) C_i^{cb} of a callback cb_i is the maximum time needed to execute it. For each callback, the set of publisher topics is defined as pt_i. The callback publishes one message to each topic in the set pt_i at the end of its execution. Furthermore, callbacks include a set of write labels wl_i and a set of read labels rl_i. Labels represent the node variables that are written to and read from during the callback's execution. We denote the WCET of a task τ_i as C_i^{τ}, which includes the WCET of its callback C_i^{cb}, the time to read data C_i^{in} and the time to write data C_i^{out}, which is further detailed in Section IV.

A *timer* is defined as a tuple $tmr_i = (T_i, cb_i)$. The period T_i is the minimum time between two activations of the timer.

A *subscription* is defined as a tuple $sub_i = (st_i, K_i, cb_i)$. It receives messages published to the subscription topic st_i. In ROS 2, each subscription only subscribes to one topic. Each subscription stores these messages in its own buffer with a maximum capacity of K_i. The subscription buffer is a FIFO queue, i.e., messages are stored in the order they are received. When the callback cb_i is executed, the oldest message from its buffer is read and then removed. If an additional message is received when the buffer is full, the oldest message in the buffer is discarded — a so-called buffer overflow.

To allow tracking of the data propagation through the system, the source of the data used for each callback has to be determined. Therefore, for DDS communication, we assume that there is exactly one publisher for each topic to ensure that all processed messages originate from the same publisher. For the communication via labels, we assume that there is at most one callback that writes to a label to ensure that the label data always originates from the same callback.

III. ROS 2 Scheduling

ROS 2 uses a scheduler abstraction, called *executor*, on top of the operating-system scheduler to manage task execution. A system may include one or more executors, and each node (and, therefore, each task) is assigned to exactly one executor. We denote the set of executors as $\mathcal{X} = \{e_1, \ldots, e_{|\mathcal{X}|}\}$.

The executors manage the order in which the assigned tasks are executed. We consider only single-threaded executors, i.e., each executor executes at most one task at a time. Furthermore, we assume that each executor is assigned to a dedicated core and that there is (at least) one additional core that manages the operating system and the DDS middleware.

A. Task Prioritization

In ROS 2, each executor assigns a fixed priority to each task that is assigned to it. In ROS 2 Humble, timers by default have a higher priority than subscriptions. Additionally, the priority of two tasks of the same type (i.e., timers or subscriptions) is determined by the order in which they are registered to the executor. Hence, there are no two tasks with the same priority on the same executor.

We define a binary priority variable $\pi_{i,j}$ that compares the priority of two tasks τ_i and τ_j. In particular, it is 1 if (i) τ_i and τ_j are assigned to the same executor and (ii) τ_i has a higher priority than τ_j; otherwise, it is 0.

B. ROS 2 Executor Scheduling

An executor's scheduling mechanism is split into two iteratively repeated phases, *polling points* and *processing windows*.

At each polling point, the executor samples one job from each activated task. A subscription is activated if its buffer is not empty. Each timer has a binary activation status that is either active or inactive. After system startup, the activation status is set to active once the timer's period elapses. Afterward, the activation status is set to active again at every integer multiple of its period, regardless of whether the previous activation has been processed or not. The activation status is set to inactive once it is sampled at a polling point. Note that timers in ROS 2 do not follow the periodic release pattern widely assumed in real-time systems research. Rather, timers are periodically *activated*, and the *release* depends on the activation status and the occurrence of polling points. A timer with a period of zero is always active; thus, the executor samples one job of such a timer at every polling point.

Immediately after a polling point, the corresponding processing window starts, where all jobs sampled at the polling point are executed non-preemptively according to their priority. We refer to the job of task τ_i in the ρ-th processing window as $J_{i,\rho}$ and to the ρ-th polling point as P_ρ. Our indexing emphasizes the processing windows (i.e., all jobs in the ρ-th processing window are indexed as $J_{*,\rho}$), but as a result, for a specific task, jobs are not necessarily indexed consecutively (e.g., there are no jobs $J_{1,1}$ and $J_{1,3}$ of task τ_1 in Figure 3).

For each job $J_{i,\rho}$ of a task τ_i in a processing window ρ, we denote the start and finish time of the job as $s_{i,\rho}$ and $f_{i,\rho}$, respectively. As the execution of tasks is non-preemptive, $f_{i,\rho} \leq s_{j,\rho} + C_i^\tau$ for all tasks τ_j. The processing window ends once all jobs sampled at the polling point are finished — at this point in time, the next polling point occurs. If no tasks assigned to an executor are active at a polling point, the executor remains idle until a task is activated, at which point the executor samples again from all activated tasks.

Our executor model is an abstraction of a real system, assuming a polling point requires no time. According to this model, infinitely many polling points may occur at the same time. Hence, we additionally assume that only finitely many polling points occur in any bounded time interval (since two polling points are separated by at least one processor cycle).

IV. Latencies

In ROS 2, the data propagation involves the time for data to be read, processed, and written by tasks, as well as additional time if the data is propagated over the network by the DDS.

For the data propagation through a task τ_i, we distinguish three types of latencies.

1) The *execution latency* is the time for executing the callback cb_i of task τ_i.
2) The *read latency* is the time for reading data from labels and subscription buffers for each job $J_{i,\rho}$ of task τ_i.
3) The *write latency* is the time for writing data to labels and subscription buffers for each job $J_{i,\rho}$ of task τ_i.

For each task τ_i, the worst-case execution time (WCET) C_i^τ is the sum of the WCET of the read latency C_i^{in}, the WCET of the execution latency C_i^{cb}, and the WCET of the write latency C_i^{out} for which the executor thread is occupied. That is,

$$C_i^\tau = C_i^{in} + C_i^{cb} + C_i^{out} \tag{1}$$

We assume that tasks read data at the beginning of a job's execution and write data at the end of a job's execution.

The read latency is the time for reading data from labels and subscription buffers. Each job reads the data from the labels in rl_i, and, if the job originates from a subscription, reads and removes the oldest message in the subscription buffer. Specifically, the read latency for subscription buffers only includes the time to access the oldest message in the buffer; not the time since the message was added to the buffer.

The write latency is the time for writing data to labels and for publishing messages to subscription.

For labels, the callback writes to the node variables referenced by the labels in wl_i. These node variables are shared between all callbacks of the node. We denote the maximum time to write to each label $\ell \in wl_i$ as $\delta_{i,\ell}^{label}$.

Publishing messages to subscription buffers involves the DDS publish-subscribe architecture by writing to the subscription buffers of the subscriptions sub_j that are subscribed to the publisher topics pt_i of the callback cb_i. Specifically, one message is published to each topic $t \in pt_i$. We denote the maximum time the executor thread is occupied with publishing a message from the callback cb_i to a topic $t \in pt_i$ as $\delta_{i,t}^{pub}$. Furthermore, if the DDS layer is involved, we denote the maximum time for the DDS thread to publish the message from the callback cb_i to the subscription sub_j as $\delta_{i,j}^{dds}$.

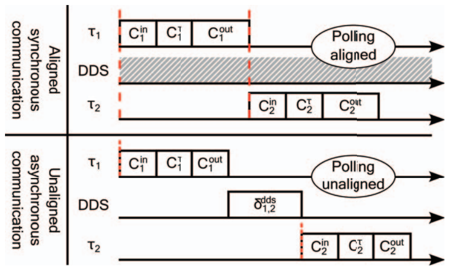

Fig. 2. Effect of task-executor assignment and DDS configuration on communication latencies (polling points indicated by red-dashed lines).

In total, the executor thread is occupied for the time C_i^{out}, which is the sum of the times for writing to labels and the times for publishing messages to subscription buffers, i.e.,

$$C_i^{out} = \sum_{\ell \in wl_i} \delta_{i,\ell}^{label} + \sum_{t \in pt_i} \delta_{i,t}^{pub}. \tag{2}$$

The ROS 2 configuration, specifically the task-to-executor assignment and the DDS configuration, affects the time for publishing a message to a subscription buffer (see Figure 2).

The task-to-executor assignment determines whether the publisher and subscriber of a message are assigned to the same executor. We refer to the communication between tasks that are assigned to the same executor as *aligned communication*, and to the communication between tasks that are assigned to different executors as *unaligned communication*. In ROS 2, aligned communication uses intra-process communication, whereas unaligned communication uses inter-process communication via DDS[1] (provided, for instance, by the eProsima FastDDS implementation [12]). Specifically, intra-process communication bypasses the DDS layer and writes the message directly to the subscription buffer by calling the subscription's reception function. In contrast, inter-process communication publishes the message to the subscription buffer via the DDS layer, and may involve networking overhead (e.g., for serialization and deserialization of the message). Hence, the task-to-executor assignment may significantly impact the write latency. In Figure 2, we illustrate the effect of the task-to-executor assignment using the polling points of the executor threads.

For DDS communication, the DDS configuration must be considered as well. Specifically, for each executor, the DDS can be configured to use either synchronous or asynchronous communication. For synchronous communication, only the executor thread is responsible for publishing the message to the subscription buffers. As a result, the executor thread is blocked until the message is fully written into all corresponding subscription buffers. In contrast, for asynchronous communication, the executor thread forwards the message to a separate DDS thread, which then publishes the message to the subscription buffers. Thus, the executor thread is only occupied until the message is forwarded to the DDS thread.

[1]We assume that system and DDS are configured to use intra-process communication for aligned and inter-process communication for unaligned communication, as this setting shows the best empirical performs [19].

In Figure 2, we illustrate the effect of the DDS configuration using the length of the outgoing communication latency C_i^{out} and the DDS thread latency $\delta_{i,j}^{dds}$.

Our model assumes the DDS latency values for synchronous and asynchronous communication to be known. Hence, our analysis and optimization is also based on these rather coarse-grained values. Previous work on DDS latencies, such as the one for FastDDS by Sciangula et al. [25], can be used to precisely determine the latencies for DDS publication and may also be utilized for more fine-grained optimization.

For notational convenience, with respect to a task τ_i, we define the total WCET of higher-priority tasks, denoted by $C_{i,hp}^{\tau}$, and lower-priority tasks, denoted by $C_{i,lp}^{\tau}$, as follows:

$$C_{i,hp}^{\tau} = \sum_{\tau_j \in \mathcal{T}} \pi_{j,i} \cdot C_j^{\tau} \tag{3}$$

$$C_{i,lp}^{\tau} = \sum_{\tau_j \in \mathcal{T}} \pi_{i,j} \cdot C_j^{\tau} \tag{4}$$

We define the WCET for a node n_i as the sum of the WCET of the tasks that are assigned to the node:

$$C_i^{node} = \sum_{\tau_j \text{ task of node } n_i} C_j^{\tau} \tag{5}$$

Similarly, the WCET of an executor e_i is the sum of the WCET of the nodes assigned to the executor:

$$C_i^{exe} = \sum_{n_j \text{ node of executor } e_i} C_j^{node} \tag{6}$$

V. End-to-End Data Propagation

Our goal is to analyze end-to-end timing for data propagation paths in ROS 2 systems. To this end, we first formally specify data propagation between tasks in ROS 2 using cause-effect chains. This specification is then extended to describe the data propagation between individual task instances using job chains. Afterward, we detail how end-to-end latencies, namely maximum reaction time and maximum data age, can be specified based on cause-effect chains.

A. Cause-Effect Chains

A cause-effect chain is a sequence of tasks that must be executed sequentially to provide a certain functionality. We focus on a single cause-effect chain for the simplicity of presentation. Practical systems have multiple cause-effect chains, which can be indexed and analyzed independently.

We adopt the notation for cause-effect chains for periodic and sporadic task systems by Günzel et al. [15], which is based on the event-chains of the AUTOSAR Timing Extensions [2]. A cause-effect chain $E = (\tau_1, \ldots, \tau_m)$ is a sequence of m tasks, which may be part of different nodes and executors. Thus, they may be communicating via labels or via DDS. We assume that the first task of each chain is a timer task transmitting sensor data, while the last task of each chain can either be a timer or a subscription, which outputs an effect using the actuators of the robot system.

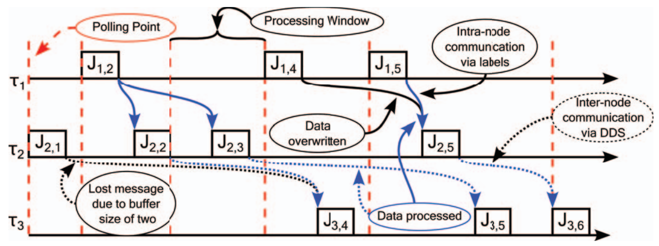

Fig. 3. Communication for the cause-effect chain $E = (\tau_1, \tau_2, \tau_3)$. Intra-node communication based on labels between τ_1 and τ_2. Inter-node communication via DDS with buffer-size 2 between τ_2 and τ_3.

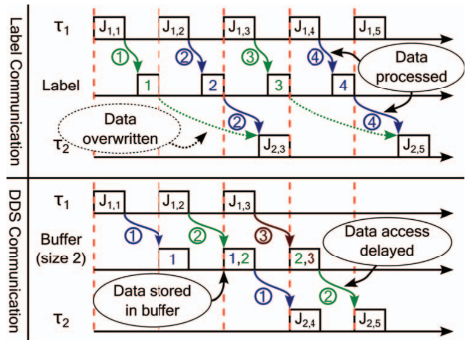

Fig. 4. Data propagation between tasks via labels or DDS.

If the task τ_{i+1} subscribes to a topic that the task τ_i publishes to (i.e., $st_{i+1} \in pt_i$), then τ_i and τ_{i+1} *communicate via DDS*. In this case, messages published by τ_i are written to the subscription buffer of τ_{i+1} (or sub_{i+1}) and processed by jobs of τ_{i+1}. Without buffer overflow, each message from τ_i triggers exactly one job of τ_{i+1}, which is executed in a future processing window. If buffer overflow occurs, messages may be discarded, which reduces the number of jobs of task τ_{i+1}.

If the task τ_{i+1} reads from a label that task τ_i writes to (i.e., $rl_{i+1} \cap wl_i \neq \emptyset$), then τ_i and τ_{i+1} *communicate via labels*. Each job of τ_i writes data to all its write labels at the end of its execution time (the so-called write-event), and each job of τ_{i+1} reads data from all read labels at the beginning of its execution time (the so-called read-event). With intra-node communication, multiple jobs of task τ_{i+1} may access the same data provided by task τ_i. Note that τ_{i+1} can even access data modified by τ_i in the same processing window if τ_{i+1} has a lower priority than τ_i.

The different communication scenarios are illustrated in Figure 3. Task τ_3 receives three messages from τ_2 before its first job $J_{3,4}$ starts executing. Due to its maximum buffer size of two, the message published by the job $J_{2,1}$ is discarded. Afterward, the jobs of τ_3 each process one message received from τ_2. The data of job $J_{1,4}$ is not processed by a job of τ_2 because the data written by $J_{1,4}$ is overwritten by $J_{1,5}$ before it can be read by $J_{2,5}$.

B. Job Chains

For subsequent tasks in a cause-effect chain, we introduce the notion of *linked* jobs for data propagation between jobs.

Definition V.1. Consider two communicating tasks τ_i and τ_{i+1}. Let J_{i,ρ_i} be a job of τ_i and let $J_{i+1,\rho_{i+1}}$ be a job of τ_{i+1}. We say that J_{i,ρ_i} *links to* $J_{i+1,\rho_{i+1}}$ if the job $J_{i+1,\rho_{i+1}}$ processes data that is written/published by the job J_{i,ρ_i} or a subsequent job $J_{i,\tilde{\rho}_i}$ with $\tilde{\rho}_i \geq \rho_i$ of task τ_i. Equivalently, we also say that $J_{i+1,\rho_{i+1}}$ *is linked to* J_{i,ρ_i}.

For communication via labels, two jobs are linked if and only if the read-event of $J_{i+1,\rho_{i+1}}$ is no earlier than the write-event of J_{i,ρ_i}. This is the same condition as used for job chains by Günzel et al. [15] (for periodic and sporadic tasks) and by Teper et al. [29] (for single-executor ROS 2 systems).

For communication via DDS, the definition of being linked takes into account the buffer architecture. In particular, even if $J_{i+1,\rho_{i+1}}$ has its read-event after the write-event of J_{i,ρ_i} it may still not be linked to J_{i,ρ_i} if the message is not first in the buffer when $J_{i+1,\rho_{i+1}}$ starts executing.

We illustrate the data propagation between two tasks τ_1 and τ_2 in Figure 4. For communication via labels, the jobs $J_{1,1}$ and $J_{2,3}$ are linked because the read-event of $J_{2,3}$ is after the write-event of $J_{1,1}$ and $J_{1,2}$. The job $J_{1,1}$ is still linked to $J_{2,1}$ even though the data is overwritten by $J_{1,2}$ before it is read by $J_{2,3}$. For communication via DDS with buffer size 2, $J_{1,2}$ and $J_{2,4}$ are not linked because the message is not first in the buffer when $J_{2,4}$ starts its execution. Rather, $J_{1,2}$ and $J_{2,5}$ are linked because the message of job $J_{1,2}$ is processed by $J_{2,5}$.

When data is available to a job J_{i,ρ_i}, we are particularly interested in two questions:

Q1. Where does this data come from?

Q2. When does this data reach the next task?

To answer Q1, we have to understand when data *was* propagated. This can be answered by our notion of being linked. More specifically, the data used by J_{i,ρ_i} comes from the *latest* job $J_{i-1,\rho_{i-1}}$ that links to J_{i,ρ_i}. As an example, we can consider job $J_{2,5}$ from Figure 3. The jobs $J_{1,2}$, $J_{1,4}$ and $J_{1,5}$ are all linked to $J_{2,5}$. The data from the latest of those jobs ($J_{1,5}$) is utilized by $J_{2,5}$.

To answer Q2, we have to examine when data *will be* propagated. This happens as soon as a job that is linked to J_{i,ρ_i} is executed. More specifically, the data reaches task τ_{i+1} at the *earliest* job $J_{i+1,\rho_{i+1}}$ that is linked to J_{i,ρ_i}. For Figure 3 job $J_{2,3}$ links to $J_{3,5}$ and all subsequent jobs of task τ_3. Hence, data from $J_{2,3}$ reaches τ_3 at job $J_{3,5}$.

This concept of data propagation is formalized by *immediate forward and immediate backward job chains*.

Definition V.2 (Immediate forward job chain)**.** Let $k \in \mathbb{N}$. We call the sequence

$$\vec{c}_k = (J_{1,\rho_1}, \dots, J_{m,\rho_m}), \tag{7}$$

of jobs an *immediate forward job chain* for cause-effect chain E, where J_{1,ρ_1} is the job of τ_1 in the k-th processing window (i.e., $\rho_1 = k$) and J_{i,ρ_i} is the *earliest* job of τ_i that is linked to $J_{i-1,\rho_{i-1}}$ for all $i = 2, \dots, m$.

We note that $\vec{c}_k$ with $k \in \mathbb{N}$ exists only if $J_{1,k}$ exists. For example, in Figure 3, the immediate forward job chain $\vec{c}_1$ does not exist because there is no job of τ_1 in the first processing window. The sequence $\vec{c}_2 = (J_{1,2}, J_{2,2}, J_{3,4})$ is an immediate forward job chain. However, $(J_{1,2}, J_{2,3}, J_{3,5})$ is not, as $J_{2,3}$ is not the earliest job of τ_2 that is linked to $J_{1,2}$.

Definition V.3 (Immediate backward job chain)**.** Let $k \in \mathbb{N}$. We call the sequence

$$\overleftarrow{c}_k = (J_{1,\rho_1}, \ldots, J_{m,\rho_m}), \tag{8}$$

of jobs an *immediate backward job chain* for cause-effect chain E, where J_{m,ρ_m} is the job of τ_m in the k-th processing window (i.e., $\rho_m = k$) and J_{i,ρ_i} is the *latest* job that is linked to $J_{i+1,\rho_{i+1}}$ for all $i = m-1, \ldots, 1$.

Again, the existence of $\overleftarrow{c}_k$ requires that there is a job $J_{m,k}$ in the k-th processing window. Moreover, it is required that for each job $J_{i+1,\rho_{i+1}}$ a linked job J_{i,ρ_i} exists.

Consider the example in Figure 3, the jobs $J_{1,4}$ and $J_{1,5}$ link to the job $J_{2,5}$, while the job $J_{2,5}$ links to the job $J_{3,6}$. Both $\vec{c}_4 = (J_{1,4}, J_{2,5}, J_{3,6})$ and $\vec{c}_5 = (J_{1,5}, J_{2,5}, J_{3,6})$ are immediate forward job chains. In contrast, only $\overleftarrow{c}_6 = (J_{1,5}, J_{2,5}, J_{3,6})$ is an immediate backward job chain, as $J_{1,4}$ is not the latest job that is linked to $J_{2,5}$. There are no immediate backward job chains $\overleftarrow{c}_k$ for $k = 1, 2, 3$ because there is no job of τ_3 in the first 3 processing windows.

C. End-To-End Latencies

In this subsection, we define the end-to-end latencies analyzed in this paper, namely the maximum reaction time and maximum data age for a specific cause-effect chain, based on the previously introduced job chains. The maximum reaction time (MRT) is the maximum latency for a cause to be propagated to an actuator. For example, it is the largest interval between a button press to lock a car's doors and them actually being locked. The maximum data age (MDA) corresponds to the maximum duration between a sensor sampling and an effect based on that sample. For example, it is the maximum length between a camera sampling and the latest time at which the steering controls are based on that sample. To achieve the formal definition of MRT and MDA, we first investigate the behavior of data propagation through the system.

Usually, we are only concerned with the system behavior when the system is properly *warmed up*. More specifically, some initial data paths that do not process any relevant data should be left out. To that end, we first make the observation that the first data arrives at the end of the first immediate forward job chain. Let $F \in \mathbb{N}$ such that $\vec{c}_F$ is the first immediate forward job chain (that exists). In Figure 3, $\vec{c}_F = (J_{1,2}, J_{2,2}, J_{3,4})$ and the first data arrives at job $J_{3,4}$.

Let $W \in \mathbb{N}$ such that the last job of $\vec{c}_F$ is $J_{m,W}$. The data processed by job $J_{m,W}$ originates from the immediate backward job chain $\overleftarrow{c}_W$. In Figure 3, we have $W = 4$ and $\overleftarrow{c}_4 = (J_{1,2}, J_{2,2}, J_{3,4})$. All jobs before $\overleftarrow{c}_W$ are considered to be part of the warm-up and are not considered for the data propagation (in Figure 3, this is only job $J_{2,1}$).

Definition V.4 (Warm-up)**.** Let $F \in \mathbb{N}$ such that $\vec{c}_F$ is the first immediate forward job chain (that exists), and let $W \in \mathbb{N}$ such that the last job of $\vec{c}_F$ is $J_{m,W}$. We denote the jobs of $\overleftarrow{c}_W$ by $\overleftarrow{c}_W = (J_{1,W_1}, \ldots, J_{m,W_m})$. All jobs of any task τ_i before J_{i,W_i}, with $i = 1 \ldots, m$, are part of the warm-up and are not further considered for the analysis.

For jobs after the warm-up, we extend the data propagation definition by including external activities (events that create data) and actuations (events that use data). In this regard, z and z' denote the start and finish of data propagation.

Immediate forward augmented job chains describe the data propagation starting from an external activity at z.

Definition V.5 (Immediate Forward Augmented Job Chain)**.** Let $z > s_{1,W_1}$ be the start of an external activity, after the start of job J_{1,W_1}, which is the first job of τ_1 after the warm-up. The immediate forward augmented job chain $\vec{ac}_z$ at z is a sequence $(z, J_{1,\rho_1}, \ldots, J_{m,\rho_m}, z')$ constructed as follows:

- J_{1,ρ_1} is the earliest job of τ_1 with $s_{1,\rho_1} \geq z$.
- $(J_{1,\rho_1}, \ldots, J_{m,\rho_m})$ is an immediate forward job chain.
- The data is processed at the finishing time of J_{m,ρ_m}. Therefore, we set $z' = f_{m,\rho_m}$.

Immediate backward augmented job chains describe where the data used for an actuation at time z' originates from.

Definition V.6 (Immediate Backward Augmented Job Chain)**.** Let $z' > f_{m,W_m}$ be the finish of an actuation, after the finish of job J_{m,W_m}, which is the first job of τ_m after the warm-up. The immediate backward augmented job chain $\overleftarrow{ac}_{z'}$ at z' is a sequence $(z, J_{1,\rho_1}, \ldots, J_{m,\rho_m}, z')$ constructed as follows:

- J_{m,ρ_m} is the latest job of τ_m with $f_{m,\rho_m} \leq z'$.
- $(J_{1,\rho_1}, \ldots, J_{m,\rho_m})$ is an immediate backward job chain.
- The data comes from job J_{1,ρ_1} started at time $z = s_{1,\rho_1}$.

The maximum reaction time is the maximal time that the system takes to fully process an external activity, whereas the maximum data age describes how old data used in an actuation is in the worst case. These metrics can be described by immediate forward/backward augmented job chains.

Definition V.7 (MRT and MDA)**.** The maximum reaction time (MRT) and the maximum data age (MDA) of a cause-effect chain E are defined as

$$MRT(E) = \sup_{z > s_{1,W_1}} len\left(\vec{ac}_z\right) \tag{9}$$

$$MDA(E) = \sup_{z' > f_{m,W_m}} len\left(\overleftarrow{ac}_{z'}\right), \tag{10}$$

respectively, where $len\left((z, J_{1,\rho_1}, \ldots, J_{m,\rho_m}, z')\right) = z' - z$ is the length of an augmented job chain.

VI. Upper Bound Analysis

In this section, we provide an upper bound on the maximum reaction time and the maximum data age for a given cause-effect chain $E = (\tau_1, \ldots, \tau_m)$. The ROS 2 system under analysis consists of multiple executors, nodes, and tasks.

We provide an upper bound on the maximum reaction time (MRT) in Theorem VI.9. To achieve this bound, we consider an arbitrary immediate forward augmented job chain $\vec{ac}_z = (z, J_{1,\rho_1}, \ldots, J_{m,\rho_m}, z')$ with $z > s_{1,W_1}$ and provide an upper bound on its length $z' - z$. Specifically, we split $z' - z$ into disjunct intervals, which we then bound individually.

For all $i = 1, \ldots, m$, we denote by t_i the earliest moment that data can be accessed by job J_{i,ρ_i}. That is, t_i is the earliest time that either (i) the message is present in the DDS buffer, or (ii) the data is written to the label that is accessed by J_{i,ρ_i}. This splits the augmented job chain into intervals related to the individual tasks, and we sum up their individual length:

$$z' - z = t_1 - z + \left(\sum_{i=1}^{m-1} t_{i+1} - t_i \right) + z' - t_m \tag{11}$$

By definition, the data is available to τ_1 at time z; thus $t_1 - z = 0$.

We further split the intervals related to each task τ_i at the start time of the job J_{i,ρ_i}, denoted by s_{i,ρ_i}, which leads to:

$$t_{i+1} - t_i = (t_{i+1} - s_{i,\rho_i}) + (s_{i,\rho_i} - t_i) \tag{12}$$

for all $i < m$, and

$$z' - t_m = (z' - s_{m,\rho_m}) + (s_{m,\rho_m} - t_m). \tag{13}$$

We provide upper bounds for $t_{i+1} - s_{i,\rho_i}$ and $z' - s_{m,\rho_m}$ in Lemma VI.1 and for $s_{i,\rho_i} - t_i$ in the subsequent lemmas.

Lemma VI.1. *Let $i = 1, \ldots, m-1$. Then $t_{i+1} - s_{i,\rho_i}$ is upper bounded by*

$$ub_i^{exe} = C_i^\tau + \delta_{i,i+1}^{dds} \tag{14}$$

if τ_i communicates with τ_{i+1} via unaligned asynchronous DDS communication, and by

$$ub_i^{exe} = C_i^\tau \tag{15}$$

otherwise. Moreover, $z' - s_{m,\rho_m}$ is upper bounded by

$$ub_m^{exe} = C_m^\tau. \tag{16}$$

Proof. Let $i \in \{1, \ldots, m-1\}$. The job J_{i,ρ_i} is executed non-preemptively for at most its WCET C_i^τ, which includes the time for reading, processing, and writing the data.

If τ_i communicates with τ_{i+1} via unaligned asynchronous DDS communication, it takes an additional latency $\delta_{i,i+1}^{dds}$ for the delay by the DDS thread until the data is transmitted (cf. Section IV). This results in $t_{i+1} - s_{i,\rho_i} \leq C_i^\tau + \delta_{i,i+1}^{dds}$.

For aligned communication and for unaligned synchronous communication, the data transmission is fully handled by the executor thread. As a result, the data is available to the next job $J_{i+1,\rho_{i+1}}$ after the job J_{i,ρ_i} finishes, as explained in Section IV. Therefore, $t_{i+1} - s_{i,\rho_i} \leq C_i^\tau$.

For the last job J_{m,ρ_m}, there is no successor in the cause-effect chain. The actuation z' occurs before or at the finish of J_{m,ρ_m}, which happens at most C_m^τ time units after the start time s_{m,ρ_m}. As a result, $z' - s_{m,\rho_m} \leq C_m^\tau$. □

We provide bounds $ub_i^{pre} \geq s_{i,\rho_i} - t_i$ in the following Lemmas, depending on the corresponding task τ_i, the parameters of τ_i, and the communication type of τ_{i-1}. In particular:

- Lemma VI.2 if τ_i is a timer and τ_i has period $T_i > 0$.
- Lemma VI.3 if τ_i is a timer and τ_i has period $T_i = 0$.
- Lemma VI.4 if τ_i is a subscription receiving a message from unaligned DDS communication.
- Lemma VI.5 if τ_i is a subscription receiving a message from aligned DDS communication.
- Lemma VI.7 if τ_i is a subscription receiving data from communication via labels.

We start with the timer tasks.

Lemma VI.2 (Timer bound non-zero period)**.** *If a timer τ_i has a non-zero period $T_i > 0$, then $s_{i,\rho_i} - t_i$ is upper bounded by:*

$$ub_i^{pre} = C_{\xi_i}^{exe} + \max\{0, T_i - C_i^\tau + C_{i,hp}^\tau\} \tag{17}$$

where $C_{\xi_i}^{exe}$ is the WCET of the executor with the index ξ_i that the task τ_i is assigned to and $C_{i,hp}^\tau$ is the total WCET of tasks with a higher priority tasks than τ_i.

Proof. We denote the last two polling points before s_{i,ρ_i} as P_{ρ_i} and $P_{\rho_i - 1}$ and consider the following cases:

Case 1: $P_{\rho_i} \leq t_i$: In that case, t_i and s_{i,ρ_i} are in the same processing window. The maximum length of the processing window is $C_{\xi_i}^{exe}$. Therefore, $s_{i,\rho_i} - t_i \leq C_{\xi_i}^{exe} \leq (17)$.

Case 2: $P_{\rho_i - 1} \leq t_i < P_{\rho_i}$: There are two subcases: For **(a)**, a job $J_{i,\rho_i - 1}$ is executed between $P_{\rho_i - 1}$ and P_{ρ_i}. Then, t_i must be after the start of $J_{i,\rho_i - 1}$, otherwise that job would be J_{i,ρ_i} and $s_{i,\rho_i} < P_{\rho_i}$. Thus, $P_{\rho_i} - t_i \leq C_i^\tau + C_{i,lp}^\tau$. For **(b)**, there is no job of τ_i between $P_{\rho_i - 1}$ and P_{ρ_i}. Then, $P_{\rho_i} - t_i \leq P_{\rho_i} - P_{\rho_i - 1} \leq C_{i,hp}^\tau + C_{i,lp}^\tau$. Combining **(a)** and **(b)** leads to $P_{\rho_i} - t_i \leq \max\{C_i^\tau, C_{i,hp}^\tau\} + C_{i,lp}^\tau$. After P_{ρ_i}, it takes at most $C_{i,hp}^\tau$ until s_{i,ρ_i}. Therefore, $s_{i,\rho_i} - t_i \leq \max\{C_i^\tau, C_{i,hp}^\tau\} + C_{i,lp}^\tau + C_{i,hp}^\tau = \max\{0, C_{i,hp}^\tau - C_i^\tau\} + C_{\xi_i}^{exe} \leq (17)$.

Case 3: $t_i < P_{\rho_{i-1}}$: The time between t_i and $P_{\rho_{i-1}}$ is at most T_i. Otherwise, the timer would have been activated before $P_{\rho_{i-1}}$, a job of τ_i would have been sampled at $P_{\rho_{i-1}}$, and $s_{i,\rho_i} < P_{\rho_i}$, which contradicts $P_{\rho_i} \leq s_{i,\rho_i}$. This leads to $P_{\rho_i} - P_{\rho_{i-1}} \leq C_{\xi_i}^{exe} - C_i^\tau$, as τ_i was not sampled at $P_{\rho_{i-1}}$. Finally, the time between P_{ρ_i} and s_{i,ρ_i} is at most $C_{i,hp}^\tau$. Therefore, $s_{i,\rho_i} - t_i \leq T_i + C_{\xi_i}^{exe} - C_i^\tau + C_{i,hp}^\tau \leq (17)$. □

Lemma VI.3 (Timer bound zero period)**.** *If a timer τ_i has a period of zero $T_i = 0$, then we derive an upper bound $ub_i^{pre} \geq s_{i,\rho_i} - t_i$ for different scenarios as follows. If τ_i does not have a predecessor task (i.e., $i = 1$) then*

$$ub_i^{pre} = C_{\xi_i}^{exe}, \tag{18}$$

where $C_{\xi_i}^{exe}$ is the WCET of the executor with the index ξ_i that the task τ_i is assigned to. If τ_i's predecessor τ_{i-1} has a higher priority than τ_i ($\pi_{i-1,i} = 1$), then

$$ub_i^{pre} = \sum_{\tau_k \in \mathcal{T} \text{ with } \pi_{i-1,k} = \pi_{k,i} = 1} C_k^\tau. \tag{19}$$

If τ_i's predecessor τ_{i-1} has a lower priority ($\pi_{i-1,i} = 0$) then

$$ub_i^{pre} = C_{i-1,lp}^\tau + C_{i,hp}^\tau. \tag{20}$$

Here, $C^{\tau}_{i,lp}$ is the total WCET of the lower priority tasks than τ_i, and $\pi_{i,j}$ is the binary task priority comparison variable.

Proof. By definition, a timer task with $T_i = 0$ is always activated, and therefore, a job is sampled at every polling point. We denote the polling point before s_{i,ρ_i} as P_{ρ_i}.

If τ_i is the first task of the chain, i.e., $i = 1$, there are two cases to consider: **Case 1:** $P_{\rho_i} \leq t_i$: In that case, t_i and s_{i,ρ_i} are in the same processing window and only jobs of tasks with higher priority than τ_i can be executed between t_i and s_{i,ρ_i}. Therefore, $s_{i,\rho_i} - t_i \leq C^{\tau}_{i,hp} <$ (18). **Case 2:** $t_i < P_{\rho_i}$: In that case, t_i must be after s_{i,ρ_i-1} since otherwise $P_{\rho_i} \leq t_i$. Hence, $P_{\rho_i} - t_i \leq P_{\rho_i} - s_{i,\rho_i-1} \leq C^{\tau}_i + C^{\tau}_{i,lp}$. Additionally, $s_{i,\rho_i} - P_{\rho_i} \leq C^{\tau}_{i,hp}$. In total, $s_{i,\rho_i} - t_i \leq C^{exe}_{\xi_i} =$ (18).

If τ_i has a predecessor task τ_{i-1}, then τ_i reads data from labels via intra-node communication, as τ_i is not a subscription. If τ_{i-1} has higher priority than τ_i (i.e., $\pi_{i-1,i} = 1$), then t_i and s_{i,ρ_i} are in the same processing window, as the job J_{i,ρ_i} of τ_i is executed after the job $J_{i-1,\rho_{i-1}}$ of τ_{i-1} in the same processing window, i.e., $\rho_i = \rho_{i-1}$. Only jobs of tasks with priority between τ_{i-1} and τ_i may be executed between t_i and s_{i,ρ_i}. That is, $\sum_{\tau_k \in \mathcal{T} \text{ with } \pi_{i-1,k}=\pi_{k,i}=1} C^{\tau}_k$.

If τ_i has a predecessor task τ_{i-1} with a lower priority than τ_i (i.e., $\pi_{i-1,i} = 0$) then t_i and s_{i,ρ_i} are in different processing windows. Hence, $P_{\rho_i} - t_i \leq C^{\tau}_{i-1,lp}$ and $s_{i,\rho_i} - P_{\rho_i} \leq C^{\tau}_{i,hp}$. In total, $s_{i,\rho_i} - t_i \leq C^{\tau}_{i-1,lp} + C^{\tau}_{i,hp} =$ (20). □

Next, we consider different scenarios for subscription tasks.

Lemma VI.4 (Subscription bound unaligned DDS communication)**.** *If τ_i is a subscription receiving messages via unaligned DDS communication, then $s_{i,\rho_i} - t_i$ is at most:*

$$ub^{pre}_i = K_i \cdot C^{exe}_{\xi_i} + \max\{0, C^{\tau}_{i,hp} - C^{\tau}_i\}, \tag{21}$$

where K_i is the subscription buffer size of τ_i and ξ_i is the index of the executor that the task τ_i is assigned to.

Proof. The message is received at time t_i. Afterward, there are at most K_i messages in the buffer. We denote the last polling point before s_{i,ρ_i} as P_{ρ_i}, and the polling point before the first job J_{i,ρ_j} of τ_i after t_i as P_{ρ_j}. There are three intervals to consider: **Interval 1:** $s_{i,\rho_i} - P_{\rho_i}$. After P_{ρ_i}, it takes at most $C^{\tau}_{i,hp}$ time units until s_{i,ρ_i}. **Interval 2:** $P_{\rho_i} - P_{\rho_j}$. It takes at most $(K_i - 1) \cdot C^{exe}_{\xi_i}$ time units for the processing windows in which all but one message in the buffer are processed. **Interval 3:** $P_{\rho_j} - t_i$. There are two cases: **Case 1:** $P_{\rho_j} \leq t_i$: Thus, $P_{\rho_j} - t_i \leq 0$. **Case 2:** $t_i < P_{\rho_j}$: Two subcases must be considered: For **(a)**, there is a job J_{i,ρ_j-1} of τ_i executing in the processing window at time t_i. Then, $s_{i,\rho_j-1} < t_i$, as otherwise, by definition of P_{ρ_j}, P_{ρ_j} must be before J_{i,ρ_j-1}, leading to $P_{\rho_j} \leq t_i$. Therefore, there are at most $C^{\tau}_i + C^{\tau}_{i,lp}$ time units between t_i and P_{ρ_j}. For **(b)**, if no job of τ_i is executed in the processing window at time t_i, then $P_{\rho_j-1} \leq t_i$; otherwise there would be a job of τ_i in the processing window after P_{ρ_j-1} which contradicts the minimality of P_{ρ_j}. Hence, the length of the processing window at t_i is at most $C^{exe}_{\xi_i} - C^{\tau}_i$. We conclude that $s_{i,\rho_i} - t_i \leq C^{\tau}_{i,hp} + (K_i - 1) \cdot C^{exe}_i + \max\{0, C^{\tau}_i + C^{\tau}_{i,lp}, C^{exe}_i - C^{\tau}_i\} =$ (21). □

Lemma VI.5 (Subscription bound aligned DDS communication)**.** *If τ_i is a subscription receiving messages from aligned DDS communication, then $s_{i,\rho_i} - t_i$ is upper bounded by:*

$$ub^{pre}_i = C^{\tau}_{i-1,lp} + C^{\tau}_{i,hp} \tag{22}$$

Proof. Teper et al. [29, Lemma 2] show that a subscription buffer that only receives messages from one task that is scheduled by the same executor never contains more than two messages, as each job that publishes a message triggers a job of the subscription, which is guaranteed to process the message in the next processing window.

After t_i, it takes at most $C^{\tau}_{i-1,lp}$ to reach the next polling point and then at most $C^{\tau}_{i,hp}$ additional time units a job of τ_{i+1} starts. Therefore, $s_{i,\rho_i} - t_i \leq C^{\tau}_{i-1,lp} + C^{\tau}_{i,hp}$. □

Since the case where subscriptions receive messages from communication via labels requires more effort, we first state the bound on the MRT for chains without such subscriptions.

Proposition VI.6 (MRT without subscription using communication via labels)**.** *If E is a cause-effect chain without subscriptions which use communication via labels, then*

$$MRT(E) \leq \sum_{i=1}^{n} ub^{pre}_i + ub^{exe}_i \tag{23}$$

is an upper bound on the maximum reaction time.

Proof. By Definition V.7, the MRT is the supremum of the length of all immediate forward augmented job chains $MRT(E) = \sup_{z > s_{1,W_1}} len\left(\vec{ac}_z\right)$ after the start of job J_{1,W_1}. Let $\vec{ac}_z$ be any immediate forward augmented job chain with $z > s_{1,W_1}$. As discussed at the beginning of this section, we can divide $z' - z$:

$$\sum_{i=1}^{n}(s_{i,\rho_i} - t_i) + \sum_{i=1}^{n-1}(t_{i+1} - s_{i,\rho_i}) + (z' - s_{n,\rho_n}) \tag{24}$$

According to Lemma VI.1, this is upper bounded by $\sum_{i=1}^{n}(s_{i,\rho_i} - t_i) + \sum_{i=1}^{n} ub^{exe}_i$ As proven by Lemmas VI.2, VI.3, VI.4, and VI.5, we can use the upper bounds ub^{pre}_i to achieve $z' - z \leq \sum_{i=1}^{n} ub^{pre}_i + ub^{exe}_i$. □

We now handle the case of subscriptions using communication via labels. In this case, the subscription is not directly triggered by the previous task τ_{i-1} but by another task τ_ψ that publishes to the subscription topic st_i of task τ_i via DDS communication. Hence, we need to account for the maximal time until τ_i is activated by τ_ψ. We denote by Δ_ψ the maximal amount of time between two activations of τ_i by τ_ψ.

Lemma VI.7 (Subscription bound label communication)**.** *If τ_i is a subscription communicating with τ_{i-1} via labels, then the following is an upper bound on $s_{i,\rho_i} - t_i$:*

$$ub^{pre}_i = \Delta_\psi + C^{exe}_{\xi_i} + \max\{0, C^{\tau}_{i,hp} - C^{\tau}_i\} \tag{25}$$

if τ_ψ communicates with τ_i via unaligned *DDS communication, and*

$$ub^{pre}_i = \Delta_\psi + C^{\tau}_{\psi,lp} + C^{\tau}_{i,hp} \tag{26}$$

if τ_ψ communicates with τ_i via aligned *DDS communication. Here, ξ_i is the index of the executor that τ_i belongs to.*

Proof. First, we prove that J_{i,ρ_i} is *not* the first job of τ_i. This proof by contradiction is achieved by comparing with the immediate backward job chain $\overleftarrow{c}_W = (J_{1,W_1}, \dots, J_{m,W_m})$ for the warm-up. Assume that J_{i,ρ_i} is the first job of τ_i. Then, J_{i,W_i} must be J_{i,ρ_i} or a subsequent job of τ_i. Since $\overleftarrow{c}_W$ is an immediate backward job chain, $J_{i-1,W_{i-1}}$ is $J_{i-1,\rho_{i-1}}$ or a subsequent job of τ_{i-1}. After several steps, we obtain that J_{1,W_1} is J_{1,ρ_1} or a subsequent job of τ_1. This means that $z \leq s_{1,\rho_1} \leq s_{1,W_1}$. This contradicts Definition V.5 of immediate forward augmented job chains, where $z > s_{1,W_1}$.

Thus, task τ_i must have been activated before t_i by τ_ψ, as τ_i can only be activated by τ_ψ. Hence, after t_i it takes at most Δ_ψ time units until τ_i is activated by τ_ψ.

We denote the last polling point before s_{i,ρ_i} as P_{ρ_i}. It takes at most $C^\tau_{i,hp}$ from P_{ρ_i} until s_{i,ρ_i}. For $P_{\rho_i} - t_i$, there are two cases: **Case 1:** $P_{\rho_i} \leq t_i$. Thus, $P_{\rho_i} - t_i \leq 0$. **Case 2:** $t_i < P_{\rho_i}$. Two subcases must be considered: For **(a)**, there is a job J_{i,ρ_i-1} of τ_i executing in the processing window at time t_i. Then $s_{i,\rho_i-1} < t_i$, as otherwise, by definition of P_{ρ_i}, P_{ρ_i} must be before J_{i,ρ_i-1}, leading to $P_{\rho_i} \leq t_i$. Therefore, $P_{\rho_i} - t_i \leq C^\tau_i + C^\tau_{i,lp}$. For **(b)**, if no job of τ_i is executed in the processing window at time t_i, then $P_{\rho_i-1} \leq t_i$; otherwise there would be a job of τ_i in the processing window after P_{ρ_i-1}. Hence, the length of the processing window at t_i is at most $C^{exe}_{\xi_i} - C^\tau_i$. We conclude that $s_{i,\rho_i} - t_i \leq C^\tau_{i,hp} + \max\{0, C^\tau_i + C^\tau_{i,lp}, C^{exe}_i - C^\tau_i\} = C^{exe}_i + \max\{0, C^\tau_{i,hp} - C^\tau_i\} <$ (25).

If τ_ψ communicates with τ_i via *aligned* DDS communication, then there is a polling point at most $C^\tau_{\psi,lp}$ time units after activation and a job of τ_i starts at most $C^\tau_{i,hp}$ time units after the polling point. Thus, $s_{i,\rho_i} - t_i \leq \Delta_\psi + C^\tau_{\psi,lp} + C^\tau_{i,hp}$. ☐

We are left with quantifying Δ_ψ. To achieve this, we construct a chain $\hat{E} = (\tau_{\psi_1}, \dots, \tau_{\psi_{\hat{m}}} = \tau_\psi)$ where τ_{ψ_1} is a timer and the tasks τ_{ψ_i} and $\tau_{\psi_{i+1}}$ communicate via DDS communication. Such a chain always exists by assumption, as stated at the end of Section II. Note that the chain $\hat{E}$ may not be a full chain of the system, but only a sub-chain, as there can be multiple timers in a chain, and we refer to τ_{ψ_1} as the last timer before the subscription task τ_i under analysis.

Lemma VI.8 (Quantify Δ_ψ)**.** *The maximal amount of time between two activations of τ_i by τ_ψ is upper bounded by:*

$$\begin{aligned}\Delta_\psi \leq \delta^{dds}_{\psi,i} + \sum_{\tau_{\psi_i} \in \hat{E}} ub^{pre}_{\psi_i} + ub^{exe}_{\psi_i} \\ - \sum_{\tau_{\psi_i} \in \hat{E}'} (K_{\psi_i} - 1) \cdot C^{exe}_{\xi_{\psi_i}}\end{aligned} \tag{27}$$

if τ_ψ communicates with τ_i via unaligned asynchronous *DDS communication, and*

$$\begin{aligned}\Delta_\psi \leq \sum_{\tau_{\psi_i} \in \hat{E}} ub^{pre}_{\psi_i} + ub^{exe}_{\psi_i} \\ - \sum_{\tau_{\psi_i} \in \hat{E}'} (K_{\psi_i} - 1) \cdot C^{exe}_{\xi_{\psi_i}}\end{aligned} \tag{28}$$

if τ_ψ communicates with τ_i via unaligned synchronous *or* aligned *DDS communication, where $\hat{E}'$ is the set of all τ_{ψ_i} in $\hat{E}$ that receive messages via unaligned DDS communication, and ξ_{ψ_i} is the index of the executor of task τ_{ψ_i}.*

Proof. Let ζ be a time point at which τ_ψ activates τ_i. We know that at the latest after $MRT(\hat{E})$ time units, a job of τ_ψ finishes. The activation of τ_i takes additional $\delta^{dds}_{\psi,i}$ time units, if τ_ψ communicates via *unaligned asynchronous* DDS communication, as the message is published by the DDS thread. If τ_ψ communicates via *unaligned synchronous* or *aligned* DDS communication, then the activation is guaranteed to be finished at the finish time of τ_i, as the message is published by the executor thread. Therefore, the $MRT(\hat{E})$ is sufficient for that case.

However, $MRT(\hat{E})$ is too pessimistic for the case of unaligned inter-node communication. In particular, we only need to bound the time until *any* job is processed, instead of a particular job. This requires only at most one processing window, instead of K_{ψ_i} processing windows. We subtract the length of $K_{\psi_i} - 1$ processing windows (each of length $C^{exe}_{\psi_i}$) from the upper bound stated in Proposition VI.6. ☐

Theorem VI.9. *The maximum reaction time of a cause-effect chain $E = (\tau_1, \dots, \tau_m)$ is upper bounded by*

$$MRT(E) \leq \sum_{i=1}^{m} ub^{pre}_i + ub^{exe}_i. \tag{29}$$

Proof. The proof coincides with the proof of Proposition VI.6, except that we include the upper bound for subscriptions using intra-node communication from Lemma VI.7. ☐

Recent work by Günzel et al. [16] shows that MRT and MDA are the same for a very general setup. In the following, we show that the equivalence is applicable to our ROS 2 system model; hence, our bound holds for the MDA as well.

Theorem VI.10. *The maximum data age of a cause-effect chain $E = (\tau_1, \dots, \tau_m)$ is upper bounded by*

$$MDA(E) \leq \sum_{i=1}^{m} ub^{pre}_i + ub^{exe}_i. \tag{30}$$

Proof. In [16] the equivalence between MRT and MDA was proven. Their system model assumes that each job $J_{i,\rho}$ has a certain read-event $re(J_{i,\rho})$ and write-event $we(J_{i,\rho})$ defined, such that the following properties are satisfied:

- P1 The read-events and write-events are ordered in the sense that $re(J_{i,\rho}) < we(J_{i,\rho})$ for any job $J_{i,\rho}$, and $re(J_{i,\rho_1}) < re(J_{i,\rho_2})$ and $we(J_{i,\rho_1}) < we(J_{i,\rho_2})$ for any two subsequent jobs J_{i,ρ_1} and J_{i,ρ_2} of the same task.
- P2 $\{re(J_{i,\rho}) \mid J_{i,\rho} \text{ job of } \tau_i\}$ and $\{we(J_{i,\rho}) \mid J_{i,\rho} \text{ job of } \tau_i\}$ have no accumulation point.
- P3 Let $\vec{c}_k = (J_{1,\rho_1}, \dots, J_{m,\rho_m})$ be an immediate forward job chain. Then $J_{i+1,\rho_{i+1}}$ is the first job of τ_{i+1} with read-event greater than or equal to $we(J_{i,\rho_i})$ for all $i = 1, \dots, m-1$.

P4 Let $\overleftarrow{c}_k = (J_{1,\rho_1}, \ldots, J_{m,\rho_m})$ be an immediate backward job chain. Then $J_{i-1,\rho_{i-1}}$ is the latest job of τ_{i-1} with write-event less than or equal to $re(J_{i,\rho_i})$ for all $i = 1, \ldots, m-1$.

As discussed in Section V, property P3 is not satisfied by default. Thus, we redefine read-events and write-events for each job in E such that they match the assumptions in [16]. We keep the read-event at the start of each job, i.e., $re(J_{i,\rho}) = s_{i,\rho}$.

Now consider a job $J_{i,\rho}$ with $i \in \{2, \ldots, n\}$. Then let $\Omega_{i,\rho}$ be the set of all jobs J_{i-1,ρ^*} of the previous task τ_{i-1} such that $J_{i,\rho}$ is the earliest job that is linked to J_{i-1,ρ^*}. (For the example in Figure 3, $\Omega_{2,5} = \{J_{1,4}, J_{1,5}\}$.) Let $|\Omega_{i,\rho}|$ be the number of jobs in $\Omega_{i,\rho}$. We define

$$\Psi_{i,\rho} = s_{i,\rho} - s_{i,\rho'} \tag{31}$$

if a previous job $J_{i,\rho'}, \ \rho' < \rho$ exists and

$$\Psi_{i,\rho} = s_{i,\rho} - \max_{J_{i-1,\rho^*} \in \Omega_{i,\rho}} (s_{i-1,\rho^*}) \tag{32}$$

otherwise. Now, all jobs of τ_{i-1} with write-event during $[s_{i,\rho} - \Psi_{i,\rho}, s_{i,\rho}]$ are linked to $s_{i,\rho}$ and $s_{i,\rho}$ is the first job they are linked to.

We define $\varepsilon_{i,\rho} := \Psi_{i,\rho}/(|\Omega_{i,\rho}|+1)$ and set the write-events of the jobs in $\Omega_{i,\rho}$ to

$$(s_{i,\rho} - |\Omega_{i,\rho|} \cdot \varepsilon), (s_{i,\rho} - (|\Omega_{i,\rho}|-1) \cdot \varepsilon), \ldots, (s_{i,\rho} - 1 \cdot \varepsilon), \tag{33}$$

ordered by their release time.

These read- and write-events fulfill the properties P1, P3, and P4. Moreover, property P2 is fulfilled because there are only finitely many polling points in any bounded time interval by our assumption in Section III-B. Therefore, MRT and MDA are equivalent for our ROS 2 system model, and $\sum_{i=1}^{m} ub_i^{pre} + ub_i^{exe}$ is an upper bound on the MDA. □

VII. Optimization

The maximum end-to-end latencies of a specific cause-effect chain must often respect a certain threshold to ensure system safety; thus, reducing end-to-end latencies is crucial. In this section, we explain how our analysis from Section VI can be utilized to minimize the end-to-end latency of cause-effect chains using constrained programming. We start by considering only a single cause-effect chain and discuss extensions to multiple cause-effect chains afterward.

Our analysis in Section VI is the first analysis that can be universally applied to different system configurations. In particular, the following parameters can be chosen freely:

- The **DDS communication mode** of each executor e_i can either be asynchronous or synchronous.
- The **task prioritization policy** of executor e_i may either prioritize timers over subscriptions (the default setting in ROS 2 Humble) or subscriptions over timers.
- The **node-executor assignment** for each node n_i.
- The **timer period** T_i of each timer tmr_i is a non-negative real number within a predefined interval $[T_{i,min}; T_{i,max}]$. A period of zero, i.e., $T_{i,min} = T_{i,max} = 0$, means that the timer would be executed in every processing window.

These parameters impact the analytical bound and can be optimized. The objective of the optimization is to minimize the upper bound on the end-to-end latency of the cause-effect chain $E = (\tau_1, \ldots, \tau_m)$, as defined in Theorem VI.9:

$$\text{minimize} \quad ub(E) := \sum_{i=1}^{m} ub_i^{pre} + ub_i^{exe} \tag{34}$$

In Section VIII we evaluate this optimization approach.

To apply our optimization to multiple cause-effect chains $E_1, \ldots, E_q$, the objective function can be chosen as follows:

1) Minimize the (weighted) sum of $ub(E_i)$: Instead of minimizing $ub(E)$, we can minimize the weighted sum $\sum_{i=1}^{q} \lambda_i ub(E_i)$. The weight λ_i accounts for the importance of the chain, i.e., minimizing $ub(E_i)$ of a chain with higher λ_i is more important.
2) Meet latency requirements: If the latency of each cause-effect chain must be below a certain threshold H_i, then we can formulate the objective function as follows:

$$\text{minimize} \quad \max_{i=1,\ldots,q} (H_i - ub(E_i)) \tag{35}$$

In case the optimization yields a result < 0, this means that all latency requirements can be met.

For the optimization, we further assume the following:

- Task-to-node assignment and the worst-case execution time (WCET) C_i^{cb} of each callback cb_i are fixed.
- If τ_i communicates with τ_j, we assume the communication type (i.e., communication via DDS or via labels) is fixed. Moreover, possible delays for label-based communication and DDS communication (aligned, unaligned synchronous, and unaligned asynchronous) are given.

Further optimization extensions can be achieved similarly.

VIII. Evaluation

In this section, we evaluate our upper-bound analysis and optimization for end-to-end latencies of ROS 2 cause-effect chains. We first describe the system under analysis, including the cause-effect chain, the system model, and the measurement setup. After that, we show our evaluation results.

A. Evaluation Setup

For our experiment, we ran an autonomous-driving software designed for high-speed oval racing [4] on a server with an i7-10700 CPU (8 x 2.9 GHz) and 32 GB RAM. We measured the latencies of the driving software stack using the framework of Betz et al. [5], which is based on ros2_tracing [3]. We conducted the simulation using the framework proposed by Betz et al. [6], which orchestrates the software execution and simulation on a separate server infrastructure.

We consider the main cause-effect chain of the autonomous-driving software, depicted in Figure 5, which consists of the LiDAR clustering pipeline, the tracking and prediction of detected objects, the local trajectory planner, and the controller. The chain consists of eight individual ROS 2 nodes and eleven tasks. For the optimization, we do not consider the LiDAR node, as it uses its own ROS 2 driver.

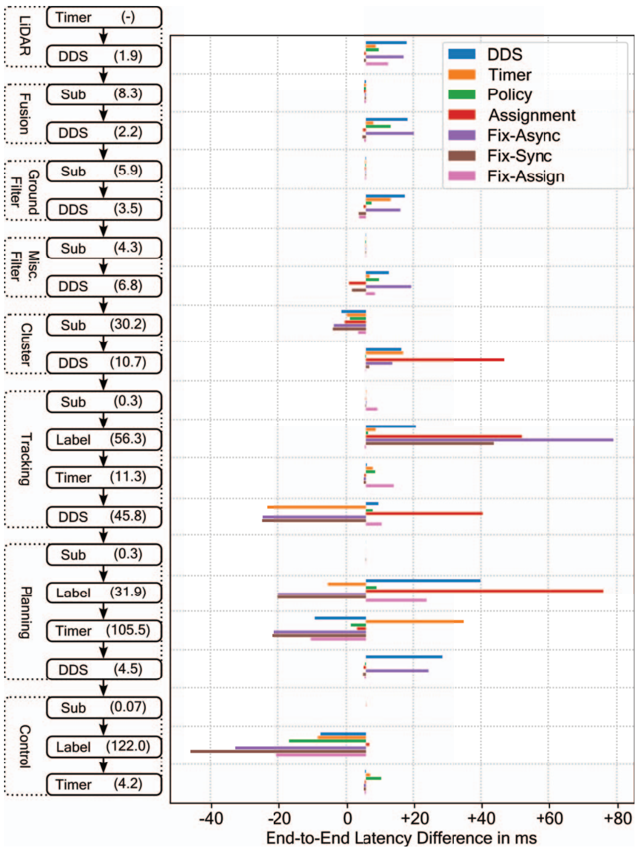

Fig. 5. End-to-end latency comparison between baseline and optimized configurations. The left side shows the nodes in dashed boxes, including the baseline latency for each step of the pipeline. The bar graph is the absolute difference in milliseconds per configuration for each step of the pipeline.

For the data propagation, the fusion, ground filter, misc. filter, and clustering nodes each have one subscription for DDS communication. The tracking, planning, and control nodes contain one subscription and a timer that sends data to the next task in the chain via DDS. The subscription and timer tasks of these three nodes communicate via labels.

For the baseline system configuration, Figure 5 shows the measured maximum WCET and DDS latencies, next to each part of the chain on the left. For the label communication latency, we specify the maximum time between the write-event of the subscription and the read-event of the timer. In our evaluation, the latencies for reading from and writing to a label, aligned DDS communication, and forwarding the message to the DDS thread are negligible and therefore not included in the figure. As shown by Betz et al. [6] and verified by our own measurements, the DDS latencies for unaligned synchronous communication by the executor thread and the publishing latency for the DDS thread are nearly the same; therefore, we only specify the former.

In our setup, the LiDAR node has a target frequency of $20\,\mathrm{Hz}$. However, due to the simulation, the output frequency can be inconsistent. Hence, to have a more realistic setup, measurements of job chains with more than the target $50\,\mathrm{ms}$ between sensor outputs are disregarded. Each run was repeated five times and, after discarding, in total approximately $70\,000$ individual latency samples were obtained per experiment.

For the subsequent optimization, certain boundary conditions had to be met. We always assign the control node to an individual executor and a fixed period of $10\,\mathrm{ms}$ to its timer, as its functionality would otherwise be compromised. Furthermore, ROS 2 does not allow assigning nodes implemented in different languages to the same executor. Thus, we restricted the node-executor assignment, as the tracking and planning nodes are implemented in Python, and all other nodes in C++.

B. Evaluation Results

For the baseline configuration, denoted as **Baseline**, the system described in Section VIII-A has one executor per node, uses the synchronous DDS mode, and prioritizes timers over subscriptions. The tracking node timer has a period of $50\,\mathrm{ms}$, and the planning node timer has a period of $75\,\mathrm{ms}$.

Table II shows, from left to right, the measured values for the mean (Mean), standard deviation (Std), 99th-percentile (P99), maximum end-to-end latencies (Max), and the upper bound using our end-to-end analysis (UB) for the baseline configuration and each optimized configuration. Figure 5 shows, for each part of the chain, the absolute difference between the maximum measured baseline latency and the maximum measured latency for each optimized configuration.

First, we describe the improvements that can be obtained by optimizing each individual parameter from Section VII.

- DDS communication mode **(DDS)**: We observed that the optimization changes it from synchronous to asynchronous. Although this increases the measured values, the upper bound is reduced. The largest latency increase occurs due to data labels not being read (see Figure 5).
- Timer period **(Timer)**: We observed that this optimization sets all timer periods in the system to $T_i = 0$. This optimization reduces both the measured data age and the analytical upper bound.
- Executor task prioritization policy **(Policy)**: We observe that the policy was changed to prioritize subscriptions over timers. This change reduces both the measured and analytical values for the system.
- Node-executor assignment **(Assignment)**: The optimization assigns the seven nodes to six executors (the fusion and ground filter nodes are assigned to one executor). This change frees one core but increases the measured values, while the upper bound remains nearly the same.

Next, we fix either the DDS communication mode or the node-executor assignment and utilize our constrained programming method to optimize the remaining parameters.

- DDS asynchronous **(Fix-Async)**: The optimization applies the same changes as **Timer** and **Policy**. For the node-executor assignment, the ground filter and misc. filter nodes are assigned to one executor. The optimization reduces the upper bound significantly, but the measured values increased compared to the baseline.

TABLE II
END-TO-END LATENCY (DATA AGE) MEASUREMENT IN ms INCLUDING THE MEAN, STANDARD DEVIATION (STD), 99TH-PERCENTILE (P99), MAXIMUM (MAX), AND UPPER BOUND (UB) FOR DIFFERENT CONFIGURATIONS.

Optimization	Mean	Std	P99	Max	UB
DDS	177.83	33.09	256.28	326.31	696.05
Timer	149.96	26.12	209.24	291.03	668.15
Policy	153.09	28.64	218.80	268.77	665.08
Assignment	171.21	34.77	259.48	353.95	832.43
Fix-Async	186.52	42.70	312.83	378.85	416.18
Fix-Sync	145.63	29.16	230.65	308.61	493.98
Fix-Assign	149.59	26.75	211.22	250.38	419.65
Baseline	156.99	32.54	244.21	312.04	835.84

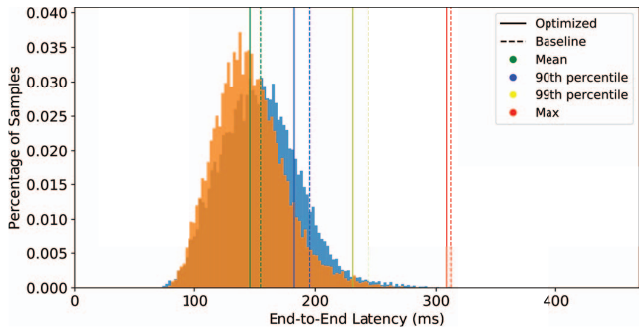

Fig. 6. Histogram of the end-to-end latency of the **Baseline** configuration (blue) and after the configuration of **Fix-Sync** is applied (orange).

- DDS synchronous **(Fix-Sync)**: The optimization applies the changes from **Timer**, **Policy**, and **Assignment**. When comparing **Fix-Sync** to **Fix-Async**, we notice that the DDS synchronous mode performs better in the measured values, but produces worse upper bounds.
- Executors-assignment **(Fix-Assign)**: The node-executor assignment is fixed, and each node is assigned to one executor. We observe that the optimization applies the same changes as **DDS**, **Timer**, and **Policy**. This configuration produces the best result for the maximum measured end-to-end latencies. However, the measured mean value is higher than for **Fix-Sync**, and the upper bound is slightly higher than for **Fix-Async**.

We conclude that our optimization can be used to reduce the measured and analytical end-to-end latency for cause-effect chains in ROS 2 systems. We achieved a reduction of at most 50.2 % for the upper bound, 19.8 % for the maximum measured end-to-end latencies, and 7.2 % for the mean measured end-to-end latencies for our system. Furthermore, we note that the analytical upper bound is not exceeded by any measured data age, which supports the correctness of our analysis.

IX. RELATED WORK

The Robot Operating System (ROS) [22], released in 2007, is a set of software libraries and tools for building robot systems. ROS enables the development of robot systems for various applications, e.g., logistics and autonomous driving, but does not consider common embedded-system constraints, like memory limitations or real-time constraints. Existing extensions of ROS that provide real-time support, such as RT-ROS [30] and ROSCH [23], are not widely adopted.

The Robot Operating System 2 (ROS 2), released in 2017, addresses these limitations and introduces real-time communication through DDS [21] as well as a new scheduler to provide possibilities for real-time guarantees. Previous work has analyzed the response time of ROS 2 systems for the ROS 2 executor using DAG-based models [7], [8] and processing chains [27]. Furthermore, novel executor designs were proposed, analyzed, and evaluated, such as PiCAS [9], the real-time executor [31], the multithreaded-executor [17], [26], and an executor for dynamic-priority scheduling [1]. For DDS communication in ROS 2, separate studies provide an analysis of the response time of for DDS message publishing [25].

For real-time systems, end-to-end timing analysis of cause-effect chains has been extensively studied. In 2009, Feiertag et al. [13] proposed the first end-to-end latency semantics that defined maximum reaction time and maximum data age. Subsequent work can be categorized into active and passive approaches. Active approaches [14], [24] specify the release of jobs in cause-effect chains to ensure the correctness of data reading and writing, while passive approaches [10], [11], [15], [18] analyze the timing behavior in an existing schedule.

In 2022, Teper et al. [29] applied end-to-end latency semantics to single-executor ROS 2 systems, considering the design of the ROS 2 architecture and its executor. Further studies by Teper et al. [28] analyzed the behavior of ROS 2 timers and subscriptions, and Betz et al. [6] studied the effects of the ROS 2 system configuration on end-to-end latencies. This paper extends the existing end-to-end timing analysis to multi-executor systems, and covers all possible communication types natively supported by ROS 2. End-to-end analysis and optimization for multi-executor ROS 2 system have, to the best of our knowledge, not yet been discussed in the literature.

X. CONCLUSION

In this paper, we present an upper-bound analysis for end-to-end latencies in multi-executor ROS 2 systems. Our analysis considers all possible communication types natively supported by ROS 2, including intra-process communication, inter-process communication, and DDS communication. Furthermore, we introduce an optimization using constrained programming that determines the optimal system configuration to minimize the upper bound on the end-to-end latencies of cause-effect chains. We evaluated our approach using an autonomous driving software stack for high-speed racing, verifying the calculated upper bounds, and determining if the configuration positively affects the end-to-end latencies of the real system. With our optimization approach, we could reduce the end-to-end latency by up to 50.2 % for the upper bound, 19.8 % for the maximum measured end-to-end latencies, and 7.2 % for the mean measured end-to-end latencies. In conclusion, our work helps to design safe and predictable real-world ROS 2 systems and improve their end-to-end latencies.

XI. Acknowledgements

This work has received funding by the German Federal Ministry of Education and Research (BMBF) in the course of the 6GEM research hub under grant number 16KISK038. This result is part of a project (PropRT) that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 865170).

References

[1] A. A. Arafat, S. Vaidhun, K. M. Wilson, J. Sun, and Z. Guo. Response time analysis for dynamic priority scheduling in ROS2. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, page 301–306, 2022.

[2] AUTOSAR. Specification of timing extensions, November 2020. release R20-11.

[3] C. Bédard, I. Lütkebohle, and M. Dagenais. ros2_tracing: Multipurpose low-overhead framework for real-time tracing of ros 2. *IEEE Robotics and Automation Letters*, 7(3):6511–6518, 2022.

[4] J. Betz, T. Betz, F. Fent, M. Geisslinger, A. Heilmeier, L. Hermansdorfer, T. Herrmann, S. Huch, P. Karle, M. Lienkamp, et al. TUM autonomous motorsport: An autonomous racing software for the Indy Autonomous Challenge. *Journal of Field Robotics*, 40(4):783–809, 2023.

[5] T. Betz, M. Schmeller, A. Korb, and J. Betz. Latency Measurement for Autonomous Driving Software Using Data Flow Extraction. In *2023 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–8, 2023.

[6] T. Betz, M. Schmeller, H. Teper, and J. Betz. How Fast is My Software? Latency Evaluation for a ROS 2 Autonomous Driving Software. In *2023 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–6, 2023.

[7] T. Blass, D. Casini, S. Bozhko, and B. B. Brandenburg. A ROS 2 Response-Time Analysis Exploiting Starvation Freedom and Execution-Time Variance. In *Proceedings of the 42nd Real-Time Systems Symposium (RTSS)*, 2021.

[8] D. Casini, T. Blass, I. Lütkebohle, and B. B. Brandenburg. Response-Time Analysis of ROS 2 Processing Chains Under Reservation-Based Scheduling. In *Proceedings of the 31st Euromicro Conference on Real-Time Systems (ECRTS)*, 2019.

[9] H. Choi, Y. Xiang, and H. Kim. PiCAS: New Design of Priority-Driven Chain-Aware Scheduling for ROS2. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 251–263, 2021.

[10] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli. Period Optimization for Hard Real-Time Distributed Automotive Systems. In *Proceedings of the 44th Annual Design Automation Conference*, page 278–283, 2007.

[11] M. Dürr, G. von der Brüggen, K.-H. Chen, and J.-J. Chen. End-to-End Timing Analysis of Sporadic Cause-Effect Chains in Distributed Systems. *ACM Trans. Embed. Comput. Syst.*, 18(5s), 2019.

[12] eProsima. eProsima Fast DDS, 2024. https://www.eprosima.com/index.php/products-all/eprosima-fast-dds.

[13] N. Feiertag, K. Richter, J. E. Nordlander, and J. Å. Jönsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *IEEE Real-Time Systems Symposium*, 2008.

[14] A. Girault, C. Prévot, S. Quinton, R. Henia, and N. Sordon. Improving and Estimating the Precision of Bounds on the Worst-Case Latency of Task Chains. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(11):2578–2589, 2018.

[15] M. Günzel, K.-H. Chen, N. Ueter, G. v. d. Brüggen, M. Dürr, and J.-J. Chen. Timing Analysis of Asynchronized Distributed Cause-Effect Chains. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 40–52, 2021.

[16] M. Günzel, H. Teper, K.-H. Chen, G. von der Brüggen, and J.-J. Chen. On the Equivalence of Maximum Reaction Time and Maximum Data Age for Cause-Effect Chains. In *Proceedings of the 35th Euromicro Conference on Real-Time Systems (ECRTS)*, 2023.

[17] X. Jiang, D. Ji, N. Guan, R. Li, Y. Tang, and Y. Wang. Real-Time Scheduling and Analysis of Processing Chains on Multi-threaded Executor in ROS 2. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 27–39, 2022.

[18] T. Kloda, A. Bertout, and Y. Sorel. Latency analysis for data chains of real-time periodic tasks. In *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 360–367, 2018.

[19] S. Macenski, A. Soragna, M. Carroll, and Z. Ge. Impact of ROS 2 Node Composition in Robotic Systems, 2023.

[20] Open Robotics. ROS 2: Humble, May 2023. https://docs.ros.org/en/humble.

[21] G. Pardo-Castellote. OMG Data-Distribution Service: architectural overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 200–206, 2003.

[22] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. volume 3, 01 2009.

[23] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio. ROSCH:Real-Time Scheduling Framework for ROS. In *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 52–58, 2018.

[24] J. Schlatow and R. Ernst. Response-Time Analysis for Task Chains in Communicating Threads. In *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–10, 2016.

[25] G. Sciangula, D. Casini, A. Biondi, C. Scordino, and M. Di Natale. Bounding the Data-Delivery Latency of DDS Messages in Real-Time Applications. In *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, 2023.

[26] H. Sobhani, H. Choi, and H. Kim. Timing Analysis and Priority-driven Enhancements of ROS 2 Multi-threaded Executors. In *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 106–118, 2023.

[27] Y. Tang, Z. Feng, N. Guan, X. Jiang, M. Lv, Q. Deng, and W. Yi. Response Time Analysis and Priority Assignment of Processing Chains on ROS2 Executors. In *2020 IEEE Real-Time Systems Symposium (RTSS)*, pages 231–243, 2020.

[28] H. Teper, T. Betz, G. von der Brüggen, K.-H. Chen, J. Betz, and J.-J. Chen. Timing-aware ros 2 architecture and system optimization. In *2023 IEEE 29th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 206–215, 2023.

[29] H. Teper, M. Günzel, N. Ueter, G. von der Brüggen, and J.-J. Chen. End-To-End Timing Analysis in ROS2. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, pages 53–65, 2022.

[30] H. Wei, Z. Shao, Z. Huang, R. Chen, Y. Guan, J. Tan, and Z. Shao. RT-ROS: A real-time ROS architecture on multi-core processors. *Future Gener. Comput. Syst.*, 56:171–178, 2016.

[31] Y. Yang and T. Azumi. Exploring Real-Time Executor on ROS 2. In *2020 IEEE International Conference on Embedded Software and Systems (ICESS)*, pages 1–8, 2020.