2020 IEEE Real-Time Systems Symposium (RTSS) | 978-1-7281-8324-4/20/$31.00 ©2020 IEEE | DOI: 10.1109/RTSS49844.2020.00030

Response Time Analysis and Priority Assignment of Processing Chains on ROS2 Executors

Yue Tang[1], Zhiwei Feng[1], Nan Guan[2], Xu Jiang[1], Mingsong Lv[2], Qingxu Deng[1], Wang Yi[1,3]

[1]Northeastern University, China
[2]The Hong Kong Polytechnic University, Hong Kong SAR
[3]Uppsala University, Sweden

***Abstract*—ROS (Robot Operating System) is currently the most popular robotic software development framework. Robotic software in safe-critical domain are usually subject to hard real-time constraints, so designers must formally model and analyze their timing behaviors to guarantee that real-time constraints are always honored at runtime. This paper studies real-time scheduling and analysis of processing chains in ROS2, the second-generation ROS with a major consideration of real-time capability. First, we study response time analysis of processing chains on ROS2 executors. We show that the only existing result of this problem is both optimistic and pessimistic, and develop new techniques to address these problems and significantly improve the analysis precision. Second, we reveal that the response time of a processing chain on an executor only depends on its last scheduling entity (*callback*), which provides useful guidance for designers to improve not only the response time bound, but also the actual worst-case/average response time of the system at little design cost. We conduct experiments with both randomly generated workload and a case study on realistic ROS2 platforms to evaluate and demonstrate our results.**

I. Introduction

ROS (Robot Operating System) [1] is currently the most popular framework and de facto standard for robotic software development. Since its version 1.0 released in 2010, ROS has been used by hundreds of thousands of developers and researchers in both industry and academia to power a tremendous number and different types of robotic systems [2], [3]. A major philosophy of ROS is to promote productivity of robotic software development by facilitating software modularity and composability, which, however, also leads to some fundamental shortcomings. A major shortcoming is lacking real-time capability, which limits wider spread of ROS as computation in robots is usually subject to timing constraints [4].

ROS2 [5], the second-generation ROS released since 2017, inherits the successful concepts of ROS and puts them onto an improved foundation [6]. A major consideration in ROS2 is to support strong real-time capability. For example, ROS2 adopts DDS (Data Distribution Service) [7] as inter-process communication middleware for real-time data exchange, can be deployed on top of real-time operating systems (instead of only running on Linux) and supports real-time-safe resource pre-allocation on control-oriented code paths.

Corresponding Author: Nan Guan, Email: nan.guan@polyu.edu.hk

The new architecture of ROS2 provides a good foundation, but on its own is insufficient to support hard real-time robotic software. For hard real-time systems, designers must verify the system timing properties and guarantee that the timing constraints are honored under any circumstance at runtime. Recently, Casini et al. [6] conducted pioneer work on formal modeling and analysis of processor scheduling in ROS2. In particular, [6] modeled the workload structure and scheduling policy in ROS2 *executors*, a core component in ROS2 multiplexing computing tasks, and developed techniques to upper-bound the response time of processing chains on ROS2 executors. It turns out that the scheduling behavior of ROS2 executors is quite different from those studied in past real-time scheduling research and requires new analysis techniques. The high-level vision and specific outcomes of [6] have made immediate impact in the ROS community [8], [9], and potentially will trigger closer interactions between the real-time scheduling and ROS communities to support analytical hard real-time guarantee for ROS2.

Despite its significance, the work in [6] has some issues: its response time bound for processing chains (called *subchains* in [6]) on a ROS2 executor is both optimistic (the bound may be smaller than the actual worst case) and pessimistic (the bound is in general unnecessarily larger than the actual worst case). We will discuss these issues in detail in Section IV.

The first contribution of this paper is addressing the above-mentioned issues in [6] by developing new response time analysis techniques for processing chains on ROS2 executors. Our new analysis techniques significantly improve the analysis precision by exploring deep insights into ROS2 executor scheduling behavior. Empirical evaluations with randomly generated workload show that our new response time bound consistently outperforms [6] with a significant margin under different parameter settings.

Our second contribution is to study how callback priority assignment influences response time of processing chains on ROS2 executors, which has not been addressed in previous work to our best knowledge. We reveal that the response time of a processing chain *only* depends on the priority of the *last* callback in this processing chain: the higher priority assigned to the last callback, the smaller response time of the processing chain potentially. This property may help ROS2 application developers to improve the system responsiveness, not only the response time upper bound for hard real-time systems, but also the actual worst-case and average response time for soft real-time and general systems, with little or even no extra design cost. We use both simulation experiments with randomly generated workload and a case study with a realistic ROS2-based robot software system to demonstrate the usage of our finding.

II. Related Work

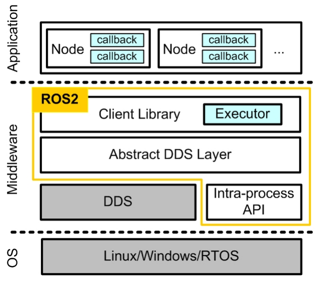
The academic research community realized the lack of real-time capability in ROS and worked on improvement on this regard. [10] introduced priority mechanisms into scheduling and data exchange of ROS and [11] ran real-time nodes of ROS on real-time operating systems for better real-time performance. [12] present a real-time scheduling framework and [13] proposed real-time ROS extension on transparent CPU/GPU coordination mechanism. However, these studies only improve the real-time system capability but not provide any analytical real-time guarantee. For ROS2, [14], [15] experimentally evaluated the performance of ROS2 with different underlying real-time operating systems and different DDS implementations, which are both measurement-based but did not consider formal modeling and analysis.

To the best of our knowledge, the only existing work on formal modeling and analysis of timing behavior of ROS2 executors is [6]. However, as mentioned above, the response time analysis for processing chains in [6] is both optimistic and pessimistic. Our work fixes these problems by exploring deeper insight of the ROS2 executor scheduling behavior and developing new analysis techniques. Moreover, we study the influence of callback priority assignment to the response time of processing chains, which was not considered in [6].

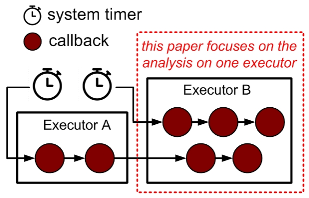
Much work has been done in real-time scheduling research on timing analysis of chain-based task models, which can be categorized into two groups according to the activation method of tasks [16]: *trigger chains* [17]–[21], where a predecessor task triggers the release of a successor task, and *data chains* [16], [22]–[25], where tasks are individually triggered and hence over- or under-sampling may occur. In the wide sense, the processing chain on ROS2 executors considered in this paper belongs to the *trigger chain* category. However, the scheduling behavior of ROS2 executors is significantly different from those studied in previous work and no existing analysis technique is applicable to our problem.

III. System Model

This section introduces the system model. We first provide some brief background about ROS2 and the context of our system model (a more detailed introduction to ROS2 can be found in [6]). Fig. 1-(a) shows the architecture of ROS2. ROS2 applications typically consist of a composition of individual *nodes* which communicate with each other using the publish-subscribe paradigm: nodes publish *messages* on *topics*, which broadcast the messages to nodes subscribed to the topic. Nodes react to incoming messages by activating *callbacks* to process



(a) ROS2 architecture



(b) Processing chains

Fig. 1. ROS2 architecture and processing chains

each message. To deploy a ROS application, the individual nodes are distributed to hosts and then mapped onto operating system processes. The *Executors* in the ROS2 client library coordinate the execution of the callbacks of the nodes in the operating system processes. ROS2 provides two built-in executors: a sequential one that executes callbacks in a single thread, and a parallel one that distributes the processing of pending callbacks across multiple threads. The same as in [6], this paper considers applications that are implemented as *processing chains* (as shown in Fig. 1-(b)) executing on *single-threaded* executors, where workload has to be executed sequentially. Each vertex in the chain represents a callback[1]. A processing chain typically starts with a *timer callback* activated by system timers, followed by several *regular callbacks* that may span over multiple executors. In this paper, we limit our attention to computing response time bound for a chain on a single executor. Nevertheless, the same as in [6], we can easily extend our results to end-to-end response time analysis for chains spanning over multiple executors by the Compositional Performance Analysis (CPA) approach [26]–[29].

A. Workload Model

We consider system Γ on a ROS2 executor, which consists of a set of independent *processing chains* (or *chains* for short) $\Gamma = \{\mathcal{C}, \mathcal{C}', \mathcal{C}''...\}$. Each chain $\mathcal{C} \in \Gamma$ consists of an ordered sequence of *callbacks* $\mathcal{C} = \{\mathcal{C}_{tm}, \mathcal{C}_1, ..., \mathcal{C}_{\|\mathcal{C}\|}\}$. $\mathcal{C}_{tm}$ is a *timer* callback, followed by *regular* callbacks $\mathcal{C}_1, ..., \mathcal{C}_{\|\mathcal{C}\|}$, where $\|\mathcal{C}\|$ represents the number of regular callbacks in $\mathcal{C}$. The last regular callback $\mathcal{C}_{\|\mathcal{C}\|}$ is called the *sink* callback of the chain. Unless explicitly specified, the convention in this paper is using $\mathcal{C} = \{\mathcal{C}_{tm}, \mathcal{C}_1, ..., \mathcal{C}_{\|\mathcal{C}\|}\}$ to denote a chain that is currently under analysis, called the *analyzed chain*, and using $\mathcal{C}' = \{\mathcal{C}'_{tm}, \mathcal{C}'_1, ..., \mathcal{C}'_{\|\mathcal{C}'\|}\}$ to denote an arbitrary other chain, called an *interfering chain*, which competes processing resource with $\mathcal{C}$. We use $e(\mathcal{C})$ to denote the total WCET (worst-case execution time) of chain $\mathcal{C}$:

$$e(\mathcal{C}) = e(\mathcal{C}_{tm}) + \sum_{z=1}^{\|\mathcal{C}\|} e(\mathcal{C}_z)$$

[1]The callbacks in a processing chain may belong to different nodes. However, this is irrelevant from the timing behavior point of view. Therefore, we will not include the node-level information in our abstract model.

where $e(\mathcal{C}_{tm})$ is the WCET of the timer callback $\mathcal{C}_{tm}$ and $e(\mathcal{C}_z)$ is the WCET of the z^{th} regular callback ($1 \leq z \leq \|\mathcal{C}\|$).

A chain releases *chain instances* repeatedly every time its timer callback receives an external event. We use $\mathcal{C}^i$ to denote the i^{th} instance of $\mathcal{C}$, which consists of the corresponding *callback instances* $\mathcal{C}^i = \{\mathcal{C}^i_{tm}, \mathcal{C}^i_1, ..., \mathcal{C}^i_{\|\mathcal{C}\|}\}$. After $\mathcal{C}^i_{tm}$ finishes execution, $\mathcal{C}^i_{tm}$ produces a message to invoke $\mathcal{C}^i_1$, and then the completion of $\mathcal{C}^i_1$ produces a message to invoke $\mathcal{C}^i_2$ and so on[2]. The release pattern of chain $\mathcal{C}$ (i.e., the arrival pattern of the external events to its timer callback) is characterized by *arrival curve* $\alpha_{\mathcal{C}}(\Delta)$, which upper-bounds the number of chain instances of $\mathcal{C}$ released in any time interval of length Δ. We use

$$\overline{\alpha_{\mathcal{C}}}(x) = \inf\{\Delta : \alpha_{\mathcal{C}}(\Delta) \geq x\}$$

to denote the pseudo-inverse function of $\alpha_{\mathcal{C}}(\Delta)$ [30], i.e., the length of any time interval in which x consecutive instances of chain $\mathcal{C}$ is released, is lower-bounded by $\overline{\alpha_{\mathcal{C}}}(x)$.

Note that our model allows $e(\mathcal{C}_{tm}) = 0$, which means that chain $\mathcal{C}$ does not have a timer callback. This is to model the case that only a part of the end-to-end processing chain of an application is allocated to the considered executor (e.g., the lower chain in Executor B in Fig. 1-(b)).

The *response time* of chain instance $\mathcal{C}^i = \{\mathcal{C}^i_{tm}, \mathcal{C}^i_1, ..., \mathcal{C}^i_{\|\mathcal{C}\|}\}$ is the time distance between its release time and the finish time of its sink callback instance $\mathcal{C}^i_{\|\mathcal{C}\|}$. The *worst-case response time* of chain $\mathcal{C}$ is the maximal response time among all its instances. The target of this paper is to calculate a safe upper bound of the worst-case response time for each chain $\mathcal{C} \in \Gamma$.

B. Resource Model

The same as [6], we assume that Γ executes on a *single-threaded executor* in ROS2 with resource reservation characterized by *supply bound function* $sbf(\Delta)$, which lower-bounds the amount of processing time available for the executor in any time interval of length Δ. We use

$$\overline{sbf}(x) = \sup\{\Delta : sbf(\Delta) < x\}$$

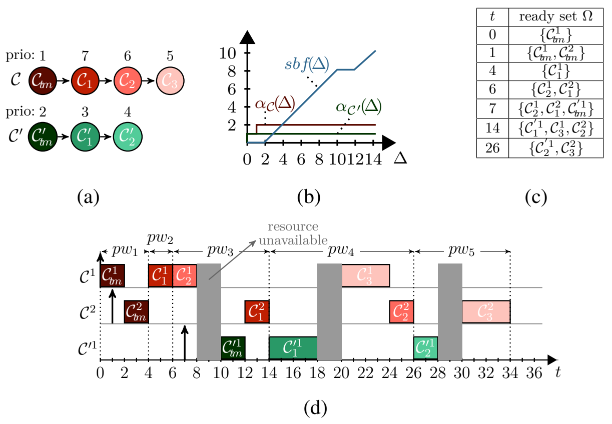
to denote the pseudo-inverse function of $sbf(\Delta)$ [30], i.e., $\overline{sbf}(x)$ upper-bounds the length of any time interval in which x units of processing time is provided.

C. Scheduling Model

The executor maintains a ready set Ω, which records *ready* callback instances to be executed. When a callback instance is finished, it is removed from Ω. A timer callback instance becomes *ready* and is put into Ω immediately as soon as it is triggered by the external event. Multiple instances of the same timer callback can exist in Ω at the same time.

A *regular* callback instance $\mathcal{C}^i_z$ is *ready* if and only if the following two conditions are both satisfied:

[2]In ROS2, a callback may produce a message in the middle of its execution. However, as we will introduce in Section III-C, the output message can only take effect at *polling points*, which cannot happen before the callback is finished. Therefore, for model simplicity, we view the output messages to be produced at the end of the callbacks.



t	ready set Ω
0	$\{\mathcal{C}^1_{tm}\}$
1	$\{\mathcal{C}^1_{tm}, \mathcal{C}^2_{tm}\}$
4	$\{\mathcal{C}^1_1\}$
6	$\{\mathcal{C}^1_2, \mathcal{C}^2_1\}$
7	$\{\mathcal{C}^1_2, \mathcal{C}^2_1, \mathcal{C}'^1_{tm}\}$
14	$\{\mathcal{C}'^1_1, \mathcal{C}^1_3, \mathcal{C}^2_2\}$
26	$\{\mathcal{C}'^1_2, \mathcal{C}^2_3\}$

(a) (b) (c) (d)

Fig. 2. An example illustrating the system model

- The input message to $\mathcal{C}^i_z$ is available, i.e., the preceding callback instance $\mathcal{C}^i_{z-1}$ (or $\mathcal{C}^i_{tm}$, if z = 1) has finished.
- All earlier instances of the same callback, $\mathcal{C}^{i-1}_z$, $\mathcal{C}^{i-2}_z$ and so on, have finished. Therefore, at most one instance of the same callback can be ready at a time.

A regular callback instance is not immediately put into Ω when it becomes ready. Instead, all currently ready regular callback instances are added to Ω at time points when Ω is empty, which are called the *polling points*.

The time interval between two polling points is called a *processing window*. The executor selects callback instances in Ω (including both timer and regular callback instances) to execute one-by-one *non-preemptively* in the current processing window. The order to execute the ready callback instances in a processing window depends on their priorities. Each callback has a fixed and unique priority and all its instances inherit this priority. The timer callbacks are on a higher priority level than regular callbacks[3]. We use $hp(\mathcal{C}_z)$ to denote the set of callbacks with higher priority than callback $\mathcal{C}_z$.

D. An Illustrating Example

Two chains $\mathcal{C}$ and $\mathcal{C}'$ in Fig. 2-(a) execute on an executor. The number above each vertex represents the priority of that callback: the smaller number the higher priority. Their arrival curves $\alpha_{\mathcal{C}}(\Delta)$ and $\alpha_{\mathcal{C}'}(\Delta)$ of the two chains and the $sbf(\Delta)$ of the executor are shown in Fig. 2-(b). Suppose the first chain instance of $\mathcal{C}$ and $\mathcal{C}'$ is released at time instant 0 and 7, respectively. The callback WCETs are: $e(\mathcal{C}_{tm}) = 2$, $e(\mathcal{C}_1) = 2$, $e(\mathcal{C}_2) = 2$, $e(\mathcal{C}_3) = 4$, $e(\mathcal{C}'_{tm}) = 2$, $e(\mathcal{C}'_2) = 4$, $e(\mathcal{C}'_3) = 2$. Fig. 2-(d) shows the execution sequence, in which the grey blocks in time interval $[8, 10]$, $[18, 20]$ and $[28, 30]$ represent that the processing resource is not available. Fig. 2-(c) shows the content of Ω at some key time points (at which some ready callback instances are added in).

[3]Regular callbacks are further categorized into several types with different priority levels, which is orthogonal to our analysis problem and thus is not modeled for presentation simplicity. We will introduce this further categorization in Section VI when discussing the priority assignment optimization.

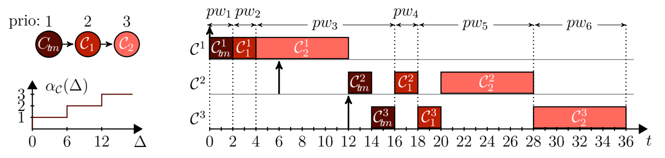


Fig. 3. An example illustrating that the analysis in [6] is optimistic

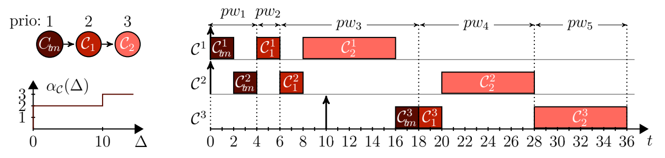


Fig. 4. An example illustrating that the analysis in [6] is pessimistic

IV. Discussions on The Analysis in [6]

In [6], the response time bound of chain $\mathcal{C}$ is computed by finding the minimum value of R satisfying (Lemma 8 in [6]):

$$sbf(R) = \sum_{\mathcal{C}' \in \Gamma} \alpha_{\mathcal{C}'}(R - e(\mathcal{C}_{\|\mathcal{C}\|}) + 1) \cdot e(\mathcal{C}') \qquad (1)$$

Note that in the above equation, $e(\mathcal{C}_{\|\mathcal{C}\|})$ is the WCET of the last regular callback of the analyzed chain $\mathcal{C}$.

We first use the example in Fig. 3 to show that the above method for computing response time bound may be too optimistic (even the system contains only one chain). We set $sbf(\Delta) = \Delta$, $e(\mathcal{C}_{tm}) = 2$, $e(\mathcal{C}_1) = 2$, $e(\mathcal{C}_2) = 8$. The arrival curve of $\mathcal{C}$ is shown in Fig. 3. We can instantiate (1) with the parameters of this example:

$$sbf(R) = \alpha_{\mathcal{C}}(R - 7) \cdot 12$$

of which $R = 12$ is the minimum solution. However, we can see in Fig. 3 that the response time of the second and third instance of $\mathcal{C}$ is 22 and 24, respectively, which are larger than the obtained bound 12.

Now we briefly discuss why the bound in [6] is too optimistic. Intuitively, the RHS of (1) computes the total workload released between a chain instance's release time and the time point when its sink callback starts execution (since a callback executes non-preemptively, workload released after this time point cannot interfere with this chain instance). As $sbf(R)$ represents the time length to provide enough resource to process such workload, solving (1) seems to safely bound the response time. However, the workload represented by the RHS of (1) only includes those released *after* the release time of the chain, but a chain instance may actually be influenced by workload released *before* its release time (the so-called "carry-in"). In our example, the second chain instance suffers "carry-in" interference by the first instance of the same chain, while the third chain instances suffers "carry-in" interference by the second instance (in general a chain instance may also suffer carry-in interference by other chains). Therefore, the RHS of (1) underestimates the total interference suffered by the analyzed chain instance as it does not count the "carry-in".

Having shown that the RHS of (1) may underestimate the interference, next we show that it may also cause overestimation. The parameters of the example in Fig. 4 are same as Fig. 3 except the arrival curve of the chain. The bound obtained by solving (1) is 36 while the actual worst-case response time is 28. Now we discuss the reason of the pessimism. Intuitively, the RHS of (1) counts all workload of every chain instance released before the starting time of the sink callback instance of the analyzed chain instance. However, actually some callback instances may execute after the sink callback instance of the analyzed chain instance, and thus do not contribute to the interference. In Fig. 4, the second chain instance has the worst-case response time, and we can see that the sink callback instance of the third chain instance actually does not interfere with the second chain instance. While in this example the overcounting is not very large, in general this may lead to significant pessimism as will be shown by the experiments in Section VII.

V. Response Time Analysis

A. Overview

We say the executor is *busy* if at least one ready callback instance has not been finished (either being executed, or waiting in the ready set Ω, or being ready but not put in Ω yet). A time interval $[a, b]$ is a *busy period* if the executor is not busy right before a and right after b, and the executor is busy at any time point in $[a, b]$.

We focus on the analysis of chain $\mathcal{C}$ in an arbitrary busy period. In particular, we focus on the analysis of $\mathcal{C}^i$, the i^{th} instance of $\mathcal{C}$ released in this busy period (i is arbitrary). We call $\mathcal{C}$ the *analyzed chain* and $\mathcal{C}^i$ the *analyzed chain instance*. All the other chains rather than $\mathcal{C}$ are *interfering chains* and their instances are *interfering chain instances*. Without loss of generality, we let time point 0 be the starting time of this busy period. We define three important time points:

t_1: the release time of $\mathcal{C}^i$,
t_2: when $\mathcal{C}^i_1$ starts execution,
t_3: when $\mathcal{C}^i_{\|\mathcal{C}\|}$ starts execution.

Our target is to find an upper bound on $t_3 - t_1$, with which we can obtain the response time bound of $\mathcal{C}^i$ by considering the time to execute $\mathcal{C}^i_{\|\mathcal{C}\|}$ (since callbacks are executed non-preemptively). In the following, we highlight some key points in our computation of the upper bound on $t_3 - t_1$.

First, we can derive an upper bound on the length of a busy period, and thus find an upper bound on the unknown index i (recall that $\mathcal{C}^i$ is an *arbitrary* instance of $\mathcal{C}$ in the considered busy period). Therefore, our analysis works as if i is a known value at each step, enumerates all values of i below its upper bound and finally gets the maximum among all values of i.

To upper-bound $t_3 - t_1$, we shall *lower*-bound t_1 and *upper*-bound t_3. Lower-bounding t_1 is easy as $\overline{\alpha_{\mathcal{C}}}(i)$ lower-bounds the length of a time interval in which $\mathcal{C}$ releases i instances.

To upper-bound t_3, we shall upper-bound the total workload executed in $[0, t_3]$. This total workload is contributed by both the analyzed chain $\mathcal{C}$ itself and the interfering chains, which will be upper-bounded respectively.

The key observation that makes our analysis more precise than [6] is as following. On ROS2 executors, a chain is executed in a "pipeline" style. Therefore, for chain instances (of either the analyzed chain $\mathcal{C}$ or an interfering chain $\mathcal{C}'$) "after" the analyzed chain instance $\mathcal{C}^i$, only part of their workload can execute before $\mathcal{C}^i_{\|\mathcal{C}\|}$ and the remaining part is excluded when counting the interference suffered by $\mathcal{C}^i$ (while in [6] all of their workload is counted as interference to $\mathcal{C}^i$).

In the following, we first introduce properties about the "pipeline"-style execution pattern of chains (Section V-B), then upper-bound the total workload of both the analyzed chain itself (Section V-C) and the interfering chains to $\mathcal{C}^i$ (Section V-D), and finally put them together and present the entire response analysis process (Section V-E).

B. Properties

We use pw_n to denote the n^{th} processing window in the considered busy period.

Lemma 1 *At most one instance of a regular callback executes in a processing window.*

Proof: At most one instance of a regular callback is ready at a time. Therefore, Ω contains at most one instance of a regular callback at the beginning of a processing window. Moreover, since new elements are added to Ω only at processing window boundaries, the lemma is proved. □

Lemma 2 *Let $\mathcal{C}$ be an arbitrary chain (the analyzed chain or an interfering chain). Suppose $\mathcal{C}^k_1$ (the first regular callback of $\mathcal{C}^k$) executes in processing window pw_n, then the earliest processing window for $\mathcal{C}^l_1$ ($l > k$) to execute is pw_{n+l-k}.*

Proof: The lemma directly follows Lemma 1. □

Lemma 3 *At most one regular callback instance of a chain instance executes in a processing window.*

Proof: When a regular callback instance is finished, its successor can be added to Ω at earliest at the end of (and thus cannot execute in) the current processing window. □

Lemma 4 *The regular callback instances of a chain instance execute in* ***consecutive*** *processing windows one by one.*

Proof: Prove by contradiction, assuming that the lemma is for the first time violated by callback instance $\mathcal{C}^i_x$, i.e., $\mathcal{C}^i_x$ does not execute in the processing window right after the one in which $\mathcal{C}^i_{x-1}$ executes (denoted by pw_n). At the end of pw_n, the input message to $\mathcal{C}^i_x$ is ready, so the only reason for $\mathcal{C}^i_x$ to not execute in pw_{n+1} is that an earlier instance $\mathcal{C}^j_x$ of the same callback executes in pw_{n+1}. Since $\mathcal{C}^i_x$ is the first callback instance violating the lemma, $\mathcal{C}^j_{x-1}$ must execute in pw_n. Therefore, we can conclude that two instances $\mathcal{C}^i_{x-1}$ and $\mathcal{C}^j_{x-1}$ of the same callback both execute in pw_n, which contradicts to Lemma 1. □

By Lemma 4 we know that the regular callback instances of the analyzed chain instance $\mathcal{C}^i$, once started, must finish in exactly $\|\mathcal{C}\|$ consecutive processing windows. Therefore, when counting other chain instances' workload in $[t_1, t_3]$, we can use this property to exclude the part of their workload that must execute after those $\|\mathcal{C}\|$ consecutive processing windows. This is the key to make our analysis more precise than [6].

C. Workload Bound of the Analyzed Chain

The following lemma gives an upper bound on the total workload of the analyzed chain $\mathcal{C}$ executed in $[0, t_3]$:

Lemma 5 *The workload of the analyzed chain $\mathcal{C}$ executed in $[0, t_3]$ is upper-bounded by $\mathcal{W}(t_3)$:*

$$\mathcal{W}(t_3) = e(\mathcal{C}) \cdot i - e(\mathcal{C}_{\|\mathcal{C}\|}) + \mathcal{M}(t_3) \tag{2}$$

$$\text{where} \quad \mathcal{M}(t_3) = \sum_{j=i+1}^{\alpha_{\mathcal{C}}(t_3)} \left(e(\mathcal{C}_{tm}) + e(\mathcal{C}_\mu) \cdot \varepsilon + \sum_{z=1}^{\mu-1} e(\mathcal{C}_z) \right)$$

$$\mu = \|\mathcal{C}\| - (j - i)$$

$$\varepsilon = \begin{cases} 1, & \mathcal{C}_\mu \in hp(\mathcal{C}_{\|\mathcal{C}\|}) \\ 0, & \textit{otherwise} \end{cases}$$

Proof: The workload of $\mathcal{C}$ in $[0, t_3]$ has two parts: (i) the workload of $\mathcal{C}^i$ itself and instances of $\mathcal{C}$ released before $\mathcal{C}^i$, (ii) the workload of instances of $\mathcal{C}$ released after $\mathcal{C}^i$. First, we upper-bound (i). The workload of $\mathcal{C}^i$ itself executed before t_3 is at most $e(\mathcal{C}) - e(\mathcal{C}_{\|\mathcal{C}\|})$. The number of instances of $\mathcal{C}$ released before $\mathcal{C}^i$ is $i - 1$, so their workload is at most $e(\mathcal{C}) \cdot (i - 1)$. Putting them together, (i) is upper-bounded by $e(\mathcal{C}) \cdot i - e(\mathcal{C}_{\|\mathcal{C}\|})$. In the remaining of the proof we upper-bound (ii).

Suppose the first regular callback instance of $\mathcal{C}^i$ executes in pw_n. By Lemma 4, we know $\mathcal{C}^i$'s regular callback instances execute in $\|\mathcal{C}\|$ consecutive processing windows $pw_n, \cdots, pw_{n+\|\mathcal{C}\|-1}$. In particular, its sink callback instance $\mathcal{C}^i_{\|\mathcal{C}\|}$ executes in $pw_{n+\|\mathcal{C}\|-1}$.

Let $\mathcal{C}^j$ be an instance of $\mathcal{C}$ released after $\mathcal{C}^i$ ($j > i$). The workload of $\mathcal{C}^j$'s timer callback instance is simply upper-bounded by $e(\mathcal{C}_{tm})$. Next we focus on bounding the workload of its regular callback instances. Let pw_m be the processing window in which $\mathcal{C}^j_1$ executes. Our goal is to analyze the workload of $\mathcal{C}^j$ executed in $pw_m, \cdots, pw_{n+\|\mathcal{C}\|-1}$ (note that t_3 is in $pw_{n+\|\mathcal{C}\|-1}$,), which consists of two parts:

1) The workload executed *before* $pw_{n+\|\mathcal{C}\|-1}$: When $m \geq n + \|\mathcal{C}\| - 1$, the workload executed before $pw_{n+\|\mathcal{C}\|-1}$ is 0. When $m < n + \|\mathcal{C}\| - 1$, by Lemma 3, at most one regular callback instance of $\mathcal{C}^j$ executes in a processing window. Therefore, the total workload of regular callback instances of $\mathcal{C}^j$ executed in $pw_m, \cdots, pw_{n+\|\mathcal{C}\|-1}$ is at most $\sum_{z=1}^{n+\|\mathcal{C}\|-1-m} e(\mathcal{C}_z)$.
2) The workload executed *in* $pw_{n+\|\mathcal{C}\|-1}$ and before t_3: When $m > n + \|\mathcal{C}\| - 1$, the workload executed in $pw_{n+\|\mathcal{C}\|-1}$ is 0. When $m \leq n + \|\mathcal{C}\| - 1$, since $\mathcal{C}^j$'s regular callback instances execute in consecutive processing windows starting from pw_m, the callback instance of $\mathcal{C}^j$ executed in $pw_{n+\|\mathcal{C}\|-1}$ is $\mathcal{C}^j_{n+\|\mathcal{C}\|-m}$, which executes before t_3 if and only if $\mathcal{C}^j_{n+\|\mathcal{C}\|-m}$ has higher priority

than $\mathcal{C}^i_{\|\mathcal{C}\|}$. Therefore, we count its workload before t_3 by $e(\mathcal{C}_{n+\|\mathcal{C}\|-m}) \cdot \varepsilon$, where $\varepsilon = 1$ if $\mathcal{C}_{n+\|\mathcal{C}\|-m} \in hp(\mathcal{C}_{\|\mathcal{C}\|})$, and $\varepsilon = 0$ otherwise.

In summary, the total workload of the regular callback instances of $\mathcal{C}^j$ executed in $[0, t_3]$ is upper-bounded by

$$e(\mathcal{C}_{n+\|\mathcal{C}\|-m}) \cdot \varepsilon + \sum_{z=1}^{n+\|\mathcal{C}\|-1-m} e(\mathcal{C}_z) \tag{3}$$

We can observe that (3) is non-increasing with respect to m. (Intuitively, if m is increased, some callbacks will be excluded from the second term of (3). Although one of them is counted in the first term of (3), the overall value of (3) cannot become larger, regardless whether ε equals 1 or 0.) Therefore, (3) is maximized when $m = n + j - i$ (by Lemma 2 we know $m \geq n + j - i$), and we can rewrite (3) with $m = n + j - i$.

$$\begin{aligned}&e(\mathcal{C}_{n+\|\mathcal{C}\|-(n+j-i)}) \cdot \varepsilon + \sum_{z=1}^{n+\|\mathcal{C}\|-1-(n+j-i)} e(\mathcal{C}_z)\\ =&e(\mathcal{C}_\mu) \cdot \varepsilon + \sum_{z=1}^{\mu-1} e(\mathcal{C}_z) \quad \text{(substitute } \|\mathcal{C}\| - (j - i) \text{ by } \mu)\end{aligned}$$

So far we have proved that the total workload of $\mathcal{C}^j$ in $[0, t_3]$ is upper-bounded by

$$e(\mathcal{C}_{tm}) + e(\mathcal{C}_\mu) \cdot \varepsilon + \sum_{z=1}^{\mu-1} e(\mathcal{C}_z) \tag{4}$$

Note that $\sum_{z=1}^{\mu-1} e(\mathcal{C}_z) = 0$ when $\mu \leq 1$ and $e(\mathcal{C}_\mu) \cdot \varepsilon = 0$ when $\mu < 1$, corresponding to the cases $m \geq n + \|\mathcal{C}\| - 1$ and $m > n + \|\mathcal{C}\| - 1$ when analyzing the workload executed before and in $pw_{n+\|\mathcal{C}\|-1}$, respectively. Since the number of instances of $\mathcal{C}$ released during $[0, t_3]$ is at most $\alpha_{\mathcal{C}}(t_3)$, summing up the upper bound in (4) for each $\mathcal{C}^j$ with $j \in [i+1, \alpha_{\mathcal{C}}(t_3)]$ proves that the workload of part (ii) is upper-bounded by $\mathcal{M}(t_3)$. □

D. Workload Bound of Interfering Chains

We use γ' to denote the number of instances of an interfering chain $\mathcal{C}'$ released in $[0, t_2]$ (i.e., released before the first regular callback instance of the analyzed chain instance $\mathcal{C}^i$ starts execution). The following lemma gives an upper bound $\mathcal{W}'(\gamma', t_3)$ on the workload of an interfering chain $\mathcal{C}'$ in $[0, t_3]$. Note that, $\mathcal{W}'(\gamma', t_3)$ depends on not only t_3, but also γ'. In the following lemma, we can treat γ' as a known value. Later we will fix the value of γ' by Lemma 7 and Lemma 8.

Lemma 6 *The workload of an interfering chain $\mathcal{C}'$ executed in $[0, t_3]$ is upper-bounded by $\mathcal{W}'(\gamma', t_3)$:*

$$\mathcal{W}'(\gamma', t_3) = \gamma' \cdot e(\mathcal{C}') + \mathcal{M}'(\gamma', t_3) \tag{5}$$

where

$$\mathcal{M}'(\gamma', t_3) = \sum_{j=\gamma'+1}^{\alpha_{\mathcal{C}'}(t_3)} \left(e(\mathcal{C}'_{tm}) + e(\mathcal{C}'_{\mu'}) \cdot \varepsilon' + \sum_{z=1}^{\min(\mu'-1, \|\mathcal{C}'\|)} e(\mathcal{C}'_z) \right)$$

$$\mu' = \|\mathcal{C}\| - (j - \gamma')$$

$$\varepsilon' = \begin{cases} 1, & \mathcal{C}'_{\mu'} \in hp(\mathcal{C}_{\|\mathcal{C}\|}) \land \|\mathcal{C}'\| > \mu' - 1 \\ 0, & \textit{otherwise} \end{cases}$$

The proof of Lemma 6 is presented in the appendix. The proof idea is similar to Lemma 5. Their difference is caused by (i) the number of regular callbacks of an interfering chain may be different from the analyzed chain (while in Lemma 5 both the analyzed chain instance and the instance contributing to the interference are from the same chain and thus have the same number of regular callbacks); (ii) the release order between an instance from the interfering chain and the analyzed chain instance may be different from the order their regular callback instances start execution (while in Lemma 5 the two considered chain instances are from the same chain and their release order and order to start executing their regular callback instances are consistent).

In the following, we will fix the value of γ'. Lemma 7 will show that $\mathcal{W}'(\gamma', t_3)$ is non-decreasing with respect to γ' and Lemma 8 will derive an upper bound on the value of γ'. Therefore, we can get an upper bound on $\mathcal{M}'(\gamma', t_3)$ by setting γ' to its upper bound.

Lemma 7 *$\mathcal{W}'(\gamma', t_3)$ is non-decreasing with respect to γ'.*

Proof: We first rewrite $\mathcal{M}'(\gamma', t_3)$ with $x = j - \gamma'$:

$$\mathcal{M}'(\gamma', t_3) = \sum_{x=1}^{\alpha_{\mathcal{C}'}(t_3)-\gamma'} \left(e(\mathcal{C}'_{tm}) + e(\mathcal{C}'_{\mu'}) \cdot \varepsilon' + \sum_{z=1}^{\min(\mu'-1, \|\mathcal{C}'\|)} e(\mathcal{C}'_z) \right)$$

$$\mu' = \|\mathcal{C}\| - x$$

$$\varepsilon' = \begin{cases} 1, & \mathcal{C}'_{\mu'} \in hp(\mathcal{C}_{\|\mathcal{C}\|}) \land \|\mathcal{C}'\| > \mu' - 1 \\ 0, & \text{otherwise} \end{cases}$$

We further define

$$\xi(x) = e(\mathcal{C}'_{tm}) + e(\mathcal{C}'_{\mu'}) \cdot \varepsilon' + \sum_{z=1}^{\min(\mu'-1, \|\mathcal{C}'\|)} e(\mathcal{C}'_z)$$

By the definition of $\xi(x)$, we can observe that $\xi(x) \leq e(\mathcal{C}')$ (intuitively, $\xi(x)$ represents the workload of callback instances of $\mathcal{C}'^{\gamma'+x}$that execute before t_3, which cannot be larger than the total WCET of the entire chain).

Given arbitrary $\gamma'_1 \leq \gamma'_2$, we have

$$\begin{aligned}&\mathcal{W}'(\gamma'_2, t_3) - \mathcal{W}'(\gamma'_1, t_3)\\ =&\gamma'_2 \cdot e(\mathcal{C}') + \sum_{x=1}^{\alpha_{\mathcal{C}'}(t_3)-\gamma'_2} \xi(x) - \left(\gamma'_1 \cdot e(\mathcal{C}') + \sum_{x=1}^{\alpha_{\mathcal{C}'}(t_3)-\gamma'_1} \xi(x) \right)\\ =&(\gamma'_2 - \gamma'_1) \cdot e(\mathcal{C}') - \left(\sum_{x=1}^{\alpha_{\mathcal{C}'}(t_3)-\gamma'_1} \xi(x) - \sum_{x=1}^{\alpha_{\mathcal{C}'}(t_3)-\gamma'_2} \xi(x) \right)\\ =&(\gamma'_2 - \gamma'_1) \cdot e(\mathcal{C}') - \sum_{x=\alpha_{\mathcal{C}'}(t_3)-\gamma'_2+1}^{\alpha_{\mathcal{C}'}(t_3)-\gamma'_1} \xi(x)\\ \geq&(\gamma'_2 - \gamma'_1) \cdot e(\mathcal{C}') - (\gamma'_2 - \gamma'_1) \cdot e(\mathcal{C}') \quad (\because \xi(x) \leq e(\mathcal{C}'))\\ =&0\end{aligned}$$

which proves the lemma. □

Lemma 8 *t_2 is upper-bounded by the minimal positive value of δ satisfying the following equation:*

$$\mathcal{X}(\delta) = sbf(\delta) \tag{6}$$

where $\mathcal{X}(\delta) = \alpha_{\mathcal{C}}(\delta)e(\mathcal{C}_{tm}) + (i-1)\sum_{z=1}^{\|\mathcal{C}\|} e(\mathcal{C}_z) + \sum_{\mathcal{C}' \in \Gamma \setminus \{\mathcal{C}\}} \alpha_{\mathcal{C}'}(\delta)e(\mathcal{C}')$

Proof: We prove the lemma by contradiction, assuming t_* is the minimal solution of (6) and $t_* < t_2$. The workload executed in $[0, t_*]$ consists of three parts:

- The workload of timer callback instances of the analyzed chain $\mathcal{C}$ released in $[0, t_*]$. The number of timer callback instances of $\mathcal{C}$ released in $[0, t_*]$ is at most $\alpha_{\mathcal{C}}(t_*)$, so their workload is upper-bounded by $\alpha_{\mathcal{C}}(t_*)e(\mathcal{C}_{tm})$.
- The workload of regular callback instances of the analyzed chain $\mathcal{C}$ that become ready in $[0, t_*]$. Since the first regular callback of $\mathcal{C}^i$ starts execution at t_2, and $t_* < t_2$, the regular callbacks instances ready in $[0, t_*]$ all belong to chain instances before $\mathcal{C}^i$. The number of such chain instances is at most $i-1$, so the corresponding workload is upper-bounded by $(i-1)\sum_{z=1}^{\|\mathcal{C}\|} e(\mathcal{C}_z)$.
- The workload of instances of interfering chains released in $[0, t_*]$. For each interfering chain $\mathcal{C}'$, the number of instances released in $[0, t_*]$ is at most $\alpha_{\mathcal{C}'}(t_*)$. Therefore, the total workload of all interfering chains' instances released in $[0, t_*]$ is upper-bounded by $\sum_{\mathcal{C}' \in \Gamma \setminus \{\mathcal{C}\}} \alpha_{\mathcal{C}'}(t_*)e(\mathcal{C}')$.

In summary, the workload that can execute in $[0, t_*]$ is upper-bounded by $\mathcal{X}(t_*)$. On the other hand, the resource available in $[0, t_*]$ is at least $sbf(t_*)$. Since $\mathcal{X}(t_*) = sbf(t_*)$, the workload that can execute in $[0, t_*]$ is at most $sbf(t_*)$, so all workload ready by t_* has been finished by t_*, which contradicts to the fact that t_* is a time point in the middle of a busy period (recall that we are analyzing the i^{th} instance of $\mathcal{C}$ in a busy period and the executor must be busy during $[0, t_3]$). ☐

We can rewrite (6) as $\delta = \overline{sbf}(\mathcal{X}(\delta))$ and apply the well-known fixed-point iteration technique [31] to find the minimal positive solution.

Combining Lemma 6, Lemma 7 and Lemma 8, we have:

Lemma 9 *The workload of an interfering chain $\mathcal{C}'$ executed in $[0, t_3]$ is upper-bounded by $\mathcal{W}'(\alpha_{\mathcal{C}'}(\overline{t_2}), t_3)$, where $\overline{t_2}$ is the minimal solution of δ in (6).*

E. Computing the Response Time Bound

So far we have obtained upper bounds for the workload of the analyzed chain and each interfering chain, and thus the total workload of the system, in $[0, t_3]$. On the other hand, $sbf(t_3)$ gives the lower bound of resource available in $[0, t_3]$. Therefore, we can upper-bound t_3 by the following lemma:

Lemma 10 *t_3 is upper-bounded by the minimal positive value of δ satisfying the following equation*

$$\mathcal{W}(\delta) + \sum_{\mathcal{C}' \in \Gamma \setminus \{\mathcal{C}\}} \mathcal{W}'(\alpha_{\mathcal{C}'}(\overline{t_2}), \delta) = sbf(\delta) \tag{7}$$

where $\overline{t_2}$ is the minimal solution of (6) in Lemma 8.

A formal proof of Lemma 10 is omitted due to space limit. Intuitively, both the LHS and RHS of (7) are non-decreasing functions with respect to δ, and their first intersection gives an upper bound on a time point by which the ready workload to be executed before the analyzed chain instance is no larger than the available resource, so the sink callback instance of the analyzed chain instance can start execution. Similar to (6), we can rewrite (7) with $\overline{sbf}$ and apply the fixed-point iteration techniques to find its minimal positive solution.

Finally, the response time bound of the analyzed chain instance $\mathcal{C}^i$ is computed by the following lemma:

Lemma 11 *The response time of $\mathcal{C}^i$ is upper-bounded by*

$$R(\mathcal{C}^i) = \overline{sbf}(sbf(\overline{t_3}) + e(\mathcal{C}_{\|\mathcal{C}\|})) - \overline{\alpha_{\mathcal{C}}}(i) \tag{8}$$

where $\overline{t_3}$ is the minimal solution of (7) in Lemma 10.

Proof: Let t_f denote the finish time of $\mathcal{C}^i_{\|\mathcal{C}\|}$, so the response time of $\mathcal{C}^i$ is $t_f - t_1$ (recall t_1 is the release time of $\mathcal{C}^i$).

The total workload in $[0, t_3]$ is upper-bounded by

$$\mathcal{W}(t_3) + \sum_{\mathcal{C}' \in \Gamma \setminus \{\mathcal{C}\}} \mathcal{W}'(\alpha_{\mathcal{C}'}(\overline{t_2}), t_3)$$

and since $t_3 \leq \overline{t_3}$, this is upper-bounded by

$$\mathcal{W}(\overline{t_3}) + \sum_{\mathcal{C}' \in \Gamma \setminus \{\mathcal{C}\}} \mathcal{W}'(\alpha_{\mathcal{C}'}(\overline{t_2}), \overline{t_3})$$

and is upper-bounded by $sbf(\overline{t_3})$ as $\overline{t_3}$ is a solution of (7).

Therefore, the total workload in $[0, t_f]$ is upper-bounded by $sbf(\overline{t_3}) + e(\mathcal{C}_{\|\mathcal{C}\|})$. Since $\overline{sbf}(x)$ upper-bounds the amount of time to finish workload x, we know

$$t_f \leq \overline{sbf}(sbf(\overline{t_3}) + e(\mathcal{C}_{\|\mathcal{C}\|})) \tag{9}$$

On the other hand, $\overline{\alpha_{\mathcal{C}}}(i)$ lower-bounds the length of a time interval in which i instances of $\mathcal{C}$ is released, so $t_1 \geq \overline{\alpha_{\mathcal{C}}}(i)$. Combining this with (9), the lemma is proved. ☐

The next lemma upper-bounds the length of a busy period, and thus upper bounds the number of instances that can be released in a busy period.

Lemma 12 *Let $\overline{\Delta}$ be the minimal positive value of Δ satisfying the following equation:*

$$\sum_{\mathcal{C}' \in \Gamma} \alpha_{\mathcal{C}'}(\Delta) \cdot e(\mathcal{C}') = sbf(\Delta) \tag{10}$$

then the number of instances released by the analyzed chain $\mathcal{C}$ in any busy period is upper-bounded by $\alpha_{\mathcal{C}}(\overline{\Delta})$.

A formal proof of the lemma is omitted due to space limit. Intuitively, the LHS of (10) upper-bounds the total workload of the entire system released in a time interval, and the RHS lower-bounds the total available resource in this time interval. The busy period must end if the total workload does not exceed the total available resource of this time interval, so the minimal solution of (10) upper-bounds the busy period length, and thus $\alpha_{\mathcal{C}}(\overline{\Delta})$ upper bounds the number of instances of $\mathcal{C}$ released in a busy period. Similar to (6) and (7), we can rewrite (10) with $\overline{sbf}$ and apply the fixed-point iteration techniques to find its minimal positive solution.

Finally, by applying Lemma 11 for each i in $[1, \alpha_{\mathcal{C}}(\overline{\Delta})]$ and getting the maximum, we get an upper bound on the worst-case response time of $\mathcal{C}$.

Theorem 1 *The worst-case response time of an arbitrary chain $\mathcal{C} \in \Gamma$ is upper-bounded by*

$$R(\mathcal{C}) = \max\{R(\mathcal{C}^i) \mid 1 \le i \le \alpha_{\mathcal{C}}(\overline{\Delta})\} \tag{11}$$

where $\overline{\Delta}$ is the minimal solution of (10) in Lemma 12.

VI. Priority Assignment

In this section we study how the priority assignment influences the response time of a chain. We observe that in the workload upper bounds of both the analyzed chain (Lemma 5) and the interfering chain (Lemma 6), the only factor that depends on the callback priorities is to check whether the callback instances executed in the same processing window as the sink callback instance $\mathcal{C}^i_{\|\mathcal{C}\|}$ of the analyzed chain instance have higher priority than $\mathcal{C}^i_{\|\mathcal{C}\|}$ or not (decided by ε in (2) and ε' in (5)). Therefore, the response time bound in Theorem 1 is only influenced by the priority of the sink callback of the analyzed chain: the higher priority, the lower response time bound potentially, but independent from the relative priority order of other callbacks. In the following, we will show that the above property holds not only for our response time bound, but also in actual system behavior.

Lemma 13 *In each processing window, the set of callback instances executed under different priority assignments is identical.*

Proof: We consider an arbitrary release sequence of the system, in which the release time of each chain instance and the actual execution time of each callback instance are fixed. The length of each busy period does not change under different priority assignments, so we limit our attention to an arbitrary busy period. First look at the first processing window pw_1 of this busy period: only timer callback instances released in pw_1 are executed in pw_1, and they all finish by the end of pw_1, so the callback instances executed in pw_1 are the same under different priority assignments (also the ready callback instances added to the ready set Ω at the polling point at the end of pw_1 are the same under different priority assignments). Next we prove that the lemma also holds for other processing windows in this busy period.

We prove by contradiction, assuming that the callback instances executed in some processing window are different under two priority assignments Φ and Φ'. We use pw_n and pw'_n to denote the nth processing window in the considered busy period under priority assignment Φ and Φ', respectively.

Let pw_m be the first processing window having such difference from its counterpart pw'_m. We assume $\mathcal{C}^i_z$ to be the earliest (in terms of release time) callback instance causing the difference, and without loss of generality, we assume that $\mathcal{C}^i_z$ executes in pw_m but not in pw'_m (the opposite case can be proved identically). We first show that $\mathcal{C}^i_z$ is not a regular callback instance. A regular callback instance $\mathcal{C}^i_z$ executes in pw_m if and only if the following conditions are both satisfied:

- Its predecessor $\mathcal{C}^i_{z-1}$ has finished by the end of pw_{m-1}
- All previous instances of the same callback $\mathcal{C}_z$ have finished by the end of pw_{m-1}.

Since pw_m is the first processing window in which the set of executed callback instances differs from its counterpart pw'_m, $\mathcal{C}^i_{z-1}$ and all previous instances of $\mathcal{C}_z$ have finished by the end of pw'_{m-1}, so $\mathcal{C}^i_z$ also executes in pw'_m. Therefore, $\mathcal{C}^i_z$ cannot be a regular callback instance, so it must be a timer callback instance (in the following we call $\mathcal{C}^i_z$ as $\mathcal{C}^i_{tm}$). We will show that this also leads to a contradiction.

We use E_* and E'_* to denote the accumulated workload executed in $pw_1, \cdots, pw_{m-1}$ and in $pw'_1, \cdots, pw'_{m-1}$, respectively. We use $\zeta(t)$ to denote the accumulated processing resource from the start time of the considered busy period to time t. Since pw_m is the first processing window in which the set of executed callback instances under Φ differs from its counterpart under Φ', we know $E_* = E'_*$.

We use t_s to denote the starting time of pw_m and t_r to denote the release time of $\mathcal{C}^i_{tm}$. We use E_1 and E'_1 to denote the total execution time of all regular callback instances executed in pw_m and pw'_m, respectively, and use E_2 to denote the total execution time of all timer callback instances that are released in $[t_s, t_r)$.

We claim that $\zeta(t_r) < E_* + E_1 + E_2$. This is because, otherwise, under priority assignment Φ, all ready callback instances should have finished before t_r, so pw_m should have ended before t_r, and thus $\mathcal{C}^i_{tm}$ cannot execute in pw_m, which contradicts to our assumption.

Now we move our attention to the execution under priority assignment Φ'. Since $\mathcal{C}^i_{tm}$ is the earliest (in terms of release time) timer callback instance that executes in pw_m but not pw'_m, we know the total workload E_2 of all timer callback instances released in $[t_s, t_r)$ are all executed before the end of pw'_m. As we discussed above, a regular callback instance that executes in pw_m must also execute in pw'_m, so $E_1 = E'_1$. Therefore, the total amount of work executed in pw'_m is at least $E'_1 + E_2 = E_1 + E_2$. And since $E_* = E'_*$, the total workload executed from the beginning of the considered busy period to the end of pw'_m is at least $E_* + E_1 + E_2$. By $\zeta(t_r) < E_* + E_1 + E_2$ we know the total amount of workload $E_* + E_1 + E_2$, which all should be finished before the end of pw'_m, has not finished by t_r. Therefore, t_r must be before the end of pw'_m. Since a timer callback instance is executed in the processing window in which it is released, $\mathcal{C}^i_{tm}$ must execute in pw'_m, which contradicts to our assumption that $\mathcal{C}^i_{tm}$ executes in pw_m but not pw'_m. $\square$

Theorem 2 *The response time of a chain instance is only affected by the priority of its sink callback (the higher priority, the lower response time potentially), but does not depend on the relative priority order of other callbacks in the system.*

Proof: We consider an arbitrary chain instance $\mathcal{C}^i$ and let pw_u be the processing window in which its sink callback instance executes. The response time of a chain instance consists of three parts: (i) the time distance between its release time and the starting time of pw_u, (ii) the time distance between the starting time of pw_u and the starting time of the sink callback instance, and (iii) the execution time of the sink callback instance. Part (iii) is clearly independent of the priority assignment. By Lemma 13, we know part (i) is independent of the priorities of callback instances executed before pw_u. The execution order of callback instances in pw_u depends on their priorities and $\mathcal{C}^i_{\|\mathcal{C}\|}$ is the only callback instance of $\mathcal{C}^i$ executing in pw_u. So part (ii) only depends on the priority of $\mathcal{C}^i$: the higher priority, the earlier it is executed potentially. The relative priority order of other callbacks in the system does not affect the starting time of $\mathcal{C}^i_{\|\mathcal{C}\|}$ in pw_u. □

Theorem 2 indicates that it is always beneficial to promote the priority of the sink callbacks in the perspective of not only the response time upper bounds (for hard real-time systems), but also actual worst-case and average response time (for soft real-time and general systems). In ROS2, the priority of a callback is decided on two levels: (1) **callback type**: Different callback types have different priority levels. As defined in our model, a timer callback has higher priority than any regular callback. Actually, regular callbacks are further classified into three types: *subscriber*, *service* and *client*. Overall, the priority order of the four callback types is

$$\text{timer} \succ \text{subscriber} \succ \text{service} \succ \text{client}$$

where $\succ$ means "has higher priority than". (2) **registration order**: The priority order of callbacks of the same type depends on the order they are registered to the executor: the earlier registered, the higher priority.

Due to the priority hierarchy mentioned above, it is not always possible to promote the priority of a sink callback to the highest. Nevertheless, the developer can grant sink callbacks as-high-as-possible priorities in the same callback type by registering them as early as possible among those of the same type, which is basically a cost-free optimization option in practice. It is also possible to perform more aggressive optimization by revising the application design. For example, as the three regular callback types mainly differ in the manner they are triggered for execution, in some applications it is possible to change the type of the callbacks (either promote the sink callback to higher-priority-level types, or demote other regular callbacks to lower-priority-level types). A thorough investigation of the gain and loss of optimization opportunities that require to revise the application is out of the scope of this paper and will be considered in our future work.

VII. Evaluations

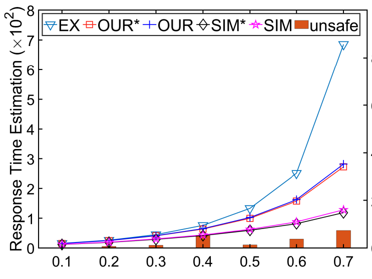
In this section, we first conduct experiments with randomly generated workload to empirically evaluate the performance improvement of our proposed response time analysis techniques and the impact to our worst-case response time bound by promoting the sink callback priority, and use a case study to demonstrate the benefit of the priority optimization to observed worst-case and average-case response time.

A. Empirical Evaluation

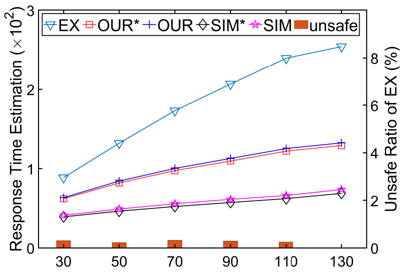
We use the TDMA model [32] for $sbf(\Delta)$ of the executor: $sbf(\Delta) = (\lfloor \Delta'/c \rfloor \cdot s + \min(\Delta' \bmod c, s)) \cdot b$, where $\Delta' = \max(\Delta - c + s, 0)$. In our experiments we set $c = 10$, $s = 8$ and $b = 1$. We randomly generate system Γ by the following method. The total utilization of the system is randomly chosen in $[0.1, 0.8]$, and the number of chains is randomly chosen in $[2, 5]$. The first callback of each chain has $1/3$ probability to be a timer callback. The arrival curve of each chain $\mathcal{C}$ is generated following the *PJD* model [27]: $\alpha_{\mathcal{C}}(\Delta) = \min(\lceil(\Delta + J)/P\rceil, \lceil \Delta/D \rceil)$, and we randomly choose P, J and D in $[60, 100]$, $[0, 2P]$ and $[1, P-1]$, respectively. Then we distribute the total utilization of the system to individual chains. The utilization of the first chain is randomly chosen from $[0.02, 2U/3]$, where U is the total utilization of the system. Then we update the remaining total utilization U by subtracting the utilization of the first chain. Then we randomly choose from $[0.02, 2U/3]$ as the utilization of the second chain and again update U by subtracting its utilization. This procedure is repeated, until only one chain is left, then we distribute all the remaining utilization to it. Next we distribute the utilization of each chain to its individual callbacks by a similar procedure as above. The order of the callbacks to get their utilization corresponds to their precedence order in the chain, and in each allocation at most 1/2 of remaining utilization is allocated. The WCET of each callback is computed by multiplying its utilization by the period P of the corresponding chain (rounded up to the nearest integer). The priority order of all callbacks in the system is randomly decided, subject to the constraint that a timer callback (the first callback of a chain) has higher priority than any regular callback. We generated 10000 systems, and for each of them we compare the response time estimations obtained by the following methods:

- OUR: the response time bound obtained by Theorem 1.
- OUR*: the same as OUR but with sink callbacks' priorities promoted to the highest among all regular callbacks of this chain (by switching its priority with the highest-priority regular callback in this chain).
- EX: the response time bound obtained by analysis in [6].
- SIM: the maximal observed response time in simulation of the system, assuming all chains release the first instances simultaneously, each chain releases instances as soon as possible and each callback instance executes to WCET. The simulation lasts until the end of a busy period. The result of SIM is a lower bound on the actual worst-case response time, but still provides useful information to evaluate the precision of our method.
- SIM*: the same as SIM but the sink callbacks' priorities promoted to the highest among all regular callbacks of this chain using the same method as OUR*.

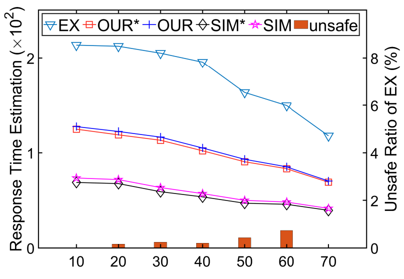
Fig. 5 shows the experiment results grouped by different parameters (the x-axis). In each figure, the five curves are the



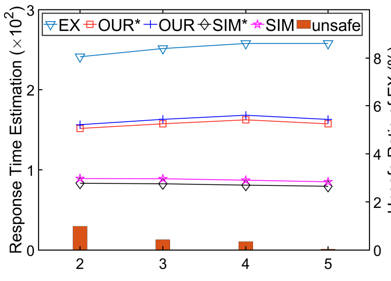
(a) x-axis: total utilization



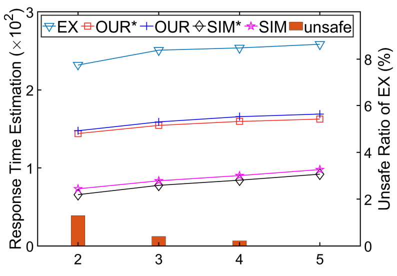
(b) x-axis: average J (for PJD arrival curve model)



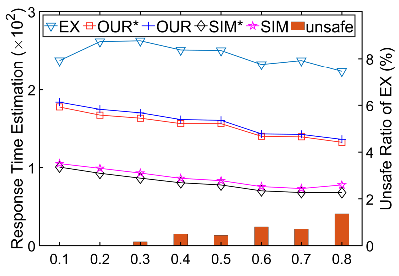
(c) x-axis: average D (for PJD arrival curve model)



(d) x-axis: number of chains



(e) x-axis: number of callbacks in each chain



(f) x-axis: average ratio of sink callback's WCET over the chain's WCET

Fig. 5. Experiment results with randomly generated workload

TABLE I
DESCRIPTION OF THE CASE STUDY

Chain	Period (ms)	Chain Description	Callback	Type	Callback Description	ACET (ms)	WCET (ms)
$\mathcal{C}$	120	Generate, publish and record dynamic joint state data	$\mathcal{C}_{\text{tm}}$	timer	Publish trigger information to $\mathcal{C}_1$	0.089	0.323
			$\mathcal{C}_1$	subscriber	Generate the original dynamic robot joint state data with random noises, pass the data through the Kalman Filter, then record and publish the joint state data	18.092	50.890
			$\mathcal{C}_2$	subscriber	Receive the joint state data, parse the given Unified Robot Description Format (URDF) file, extract the robot model, transform the joint state data, and publish joint state data	16.399	36.543
			$\mathcal{C}_3$	subscriber	Receive and record the joint state data in the file, and calculate the delay	7.054	14.585
$\mathcal{C}'$	120	Generate, publish and record laser scans	$\mathcal{C}'_{\text{tm}}$	timer	Publish trigger information to $\mathcal{C}'_1$	0.054	0.335
			$\mathcal{C}'_1$	subscriber	Generate the laser scans, pass the data through the Kalman Filter, and publish the laser scans	7.444	20.177
			$\mathcal{C}'_2$	subscriber	Receive and record laser scans in the file and calculate the delay	9.579	17.326
$\mathcal{C}''$	120	Generate, publish and record fixed joint state data	$\mathcal{C}''_{\text{tm}}$	timer	Publish trigger information to $\mathcal{C}''_1$	0.039	0.248
			$\mathcal{C}''_1$	subscriber	Generate the fixed robot joint state data with random noises, record data, then pass the data through the Kalman Filter, and publish the fixed joint state data	9.927	20.780
			$\mathcal{C}''_2$	subscriber	Receive and record the fixed joint state data in the file, and calculate the delay	9.535	13.853

TABLE II
EXPERIMENT RESULTS OF THE CASE STUDY

	$\mathcal{C}$			$\mathcal{C}'$			$\mathcal{C}''$		
Priority Assignment	I	II	III	I	II	III	I	II	III
Average (ms)	93.905	93.955	85.335	76.918	76.849	67.804	86.115	85.891	77.719
Worst-case (ms)	139.441	134.917	129.108	125.770	120.984	114.950	130.799	127.536	122.159

average response time estimation obtained by the above five methods (the values corresponding to the left y-axis). Each result on a curve is the average response time estimation of all chains of all generated systems with that specific parameter falling in range corresponding to each x-axis (except 5-(d) and (e) where x-axis values are discrete). For example, the value 0.3 on the x-axis in Fig. 5-(a) means the utilization range $[0.25, 0.35)$. In Fig. 5-(d) to (f), the results only include systems with total utilization in the range $[0.55, 0.65)$. This is because the grouping according to the x-axis of these figures has a strong correlation with the system total utilization. For example, a system with more chains typically has a higher total utilization. Therefore, we only evaluate the systems in a fixed utilization range to better reveal the result trends with respect to these parameters. In each figure, we also report the ratio of systems with which the response time estimation of at least one chain obtained by EX is smaller than SIM, represented by the histogram (the values corresponding to the right y-axis). In these cases, the results of EX must be unsafe since SIM gives a *lower* bound on the actual worst-case response time. From the

TABLE III
PRIORITY ASSIGNMENTS IN THE CASE STUDY

	$\mathcal{C}_{\text{tm}}$	$\mathcal{C}_1$	$\mathcal{C}_2$	$\mathcal{C}_3$	$\mathcal{C}'_{\text{tm}}$	$\mathcal{C}'_1$	$\mathcal{C}'_2$	$\mathcal{C}''_{\text{tm}}$	$\mathcal{C}''_1$	$\mathcal{C}''_2$
I	1	4	5	6	2	7	8	3	9	10
II	1	5	4	6	2	7	8	3	9	10
III	1	6	5	4	2	8	7	3	10	9

results in Fig. 5 we can see that the our new analysis method consistently outperforms EX with a significant margin under different parameter settings. By promoting the priority of sink callbacks, the obtained response time bounds are improved (by $5\% - 8\%$ on average).

B. Case Study

The case study consists of three processing chains deployed on a single-threaded executor in version *Eloquent Elusor* of ROS2 (released in November 22nd, 2019), running on top of Ubuntu 18.04.5 with kernel 5.4.0-48-generic and a desktop with Intel i7-9700F CPU@3.00 GHz and total memory of 16 GB (one core is used to run this ROS2 executor).

Intra-process communication is used to publish/subscribe messages among the callbacks on each chain. Each chain starts with a timer callback triggered by system timers with a fixed period. Table I gives the brief description of the chains and callbacks. The ACET and WCET columns are the average and worst-case observed execution time of each callback in $10,000$ runs.

Table II reports the average and worst-case observed response time of each chain under different priority assignments as shown in Table III. Under each priority assignment, we run the system for 20 minutes. "I" is the default priority assignment, and "II" switches the priority of two non-sink regular callbacks in $\mathcal{C}$. We can see that this priority adjustment causes no significant change in the average/worst-case observed response time. In priority assignment "III", we switch the priority of the sink callback in each chain with the highest-priority regular callback in that chain, which leads to clear improvement in both the average and worst-case observed response time.

An interesting phenomenon we observed in the case study is that, although in our model (also the model in [6]), if some message is produced in a processing window, it must be considered at the polling point at the end of this processing window, this is not always the case in reality. There are cases where the message actually cannot catch up the polling point at the end of that processing window, due to the information propagation delay in ROS. This delay is more significant when the information propagates across different cores. As such behavior deviates from our assumed model, we by purpose exclude it from the experiments reported above. To this end, on one hand, we run not only the considered executor, but also the entire ROS2 stack on a single core, to avoid inter-core information propagation, and on the other hand, we analyze the execution log of the experiments (with slightly different period and jitter settings) and make sure that the reported results do not contain such unwanted behavior. Although in this paper we do not further investigate this phenomenon in depth, this problem indeed needs to be seriously addressed for the theoretical results of this paper to be applicable in practice. For example, the analysis techniques proposed in this paper could be extended by considering the extra delay caused by missing the nearest polling point in the worst case. Actually, as this behavior may increase the end-to-end processing delay for both the average case and the worst case, and makes the system more unpredictable in timing, we believe it deserves efforts to improve the ROS architecture to eliminate this behavior, at least for real-time applications. We leave further investigation of this problem to our future work.

VIII. CONCLUSION AND FUTURE WORK

We develop response time analysis techniques for processing chains on ROS2 single-threaded executors, which not only fix the optimistic results in [6] but also significantly improve the precision. We also proved that a processing chain's response time on an executor only depends on its last callback, by which the designer can optimize the priority assignment to improve not only the response time upper bound but also the actual worst-case and average response time of the system. Our ultimate goal is to provide analytical timing guarantee for ROS-based robotic software systems. This paper is a small step towards this goal and still subject to many limitations. Next step, we will extend our work to more general and complex settings, such as multi-threaded executors, integrating the per-executor analysis with the modeling and analysis of delay incurred by communication through DDS, as well as improving the ROS architecture for better timing predictability (e.g., to avoid the problem stated at the end of Section VII-B).

APPENDIX: PROOF OF LEMMA 6

Proof: The workload of $\mathcal{C}'$ executed in $[0, t_3]$ consists of two parts: (i) the workload of instances released before or at t_2, (ii) the workload of instances released after t_2. Part (i) can be simply upper-bounded by $\gamma' \cdot e(\mathcal{C}')$. In the following, we focus on proving that part (ii) is upper-bounded by $\mathcal{M}'(\gamma', t_3)$.

Suppose the first regular callback instance of the analyzed chain instance $\mathcal{C}^i$ executes in pw_n. By Lemma 4, we know $\mathcal{C}^i$'s regular callback instances execute in $\|\mathcal{C}\|$ consecutive processing windows $pw_n, \cdots, pw_{n+\|\mathcal{C}\|-1}$. In particular, its last regular callback instance executes in $pw_{n+\|\mathcal{C}\|-1}$.

Let $\mathcal{C}'^j$ be the j^{th} instance of $\mathcal{C}'$, $j > \gamma'$. The workload of its timer callback is at most $e(\mathcal{C}'_{tm})$. Next, we focus on bounding the workload of its regular callback instances.

Let pw_m be the processing window in which $\mathcal{C}'^j_1$ executes. Our goal is to analyze the workload of $\mathcal{C}'^j$ executed in $pw_m, \cdots, pw_{n+\|\mathcal{C}\|-1}$ (note that t_3 is in $pw_{n+\|\mathcal{C}\|-1}$), which consists of two parts:

(1) The workload executed *before* $pw_{n+\|\mathcal{C}\|-1}$. We distinguish the following cases:

- $m \geq n + \|\mathcal{C}\| - 1$. In this case, $\mathcal{C}'^j$ starts execution in or after $pw_{n+\|\mathcal{C}\|-1}$, so its workload executed before $pw_{n+\|\mathcal{C}\|-1}$ is 0.

- $m < n + \|\mathcal{C}\| - 1$. By Lemma 4, the regular callback instances of $\mathcal{C}'^{j}$ execute in consecutive processing windows starting from pw_m. We further distinguish two cases:
  - $\|\mathcal{C}'\| \leq n+\|\mathcal{C}\|-1-m$. In this case, $\mathcal{C}'^{j}$ completes before $pw_{n+\|\mathcal{C}\|-1}$, so the workload of regular callbacks of $\mathcal{C}'^{j}$ executed *before* $pw_{n+\|\mathcal{C}\|-1}$ is upper-bounded by $\sum_{z=1}^{\|\mathcal{C}'\|} e(\mathcal{C}'_z)$.
  - $\|\mathcal{C}'\| > n + \|\mathcal{C}\| - 1 - m$. In this case, the last callback instance executes *before* $pw_{n+\|\mathcal{C}\|-1}$ is $\mathcal{C}'_{\|\mathcal{C}\|+n-1-m}$, so the workload of regular callbacks of $\mathcal{C}'^{j}$ executed *before* $pw_{n+\|\mathcal{C}\|-1}$ is upper-bounded by $\sum_{z=1}^{n+\|\mathcal{C}\|-1-m} e(\mathcal{C}'_z)$.

Combining these cases upper-bounds the workload of regular callback instances of $\mathcal{C}'^{j}$ executed before $pw_{n+\|\mathcal{C}\|-1}$:

$$\sum_{z=1}^{\min(n+\|\mathcal{C}\|-1-m,\|\mathcal{C}'\|)} e(\mathcal{C}'_z)$$

(2) The workload executed *in* $pw_{n+\|\mathcal{C}\|-1}$ and before t_3. Again, we distinguish two cases:

- $m > n + \|\mathcal{C}\| - 1$. In this case, $\mathcal{C}'^{j}$ starts execution after $pw_{n+\|\mathcal{C}\|-1}$, so its workload executed in $pw_{n+\|\mathcal{C}\|-1}$ is 0.
- $m \leq n + \|\mathcal{C}\| - 1$. We further distinguish two cases:
  - $\|\mathcal{C}'\| \leq \|\mathcal{C}\| + n - 1 - m$. In this case, $\mathcal{C}'^{j}$ completes before $pw_{n+\|\mathcal{C}\|-1}$, so the workload of $\mathcal{C}'^{j}$ in $pw_{n+\|\mathcal{C}\|-1}$ is 0.
  - $\|\mathcal{C}'\| > \|\mathcal{C}\| + n - 1 - m$. In this case, the regular callback instance of $\mathcal{C}'^{j}$ executed in $pw_{n+\|\mathcal{C}\|-1}$ is $\mathcal{C}'^{j}_{n+\|\mathcal{C}\|-m}$, which executes before t_3 if and only if $\mathcal{C}'^{j}_{n+\|\mathcal{C}\|-m}$ has higher priority than $\mathcal{C}^{i}_{\|\mathcal{C}\|}$.

Combining the discussion of the above cases we have proved that the workload of regular callback instance of $\mathcal{C}'^{j}_1$ in $pw_{n+\|\mathcal{C}\|-1}$ is upper-bounded by $e(\mathcal{C}'_{n+\|\mathcal{C}\|-m}) \cdot \varepsilon'$ where

$$\varepsilon' = \begin{cases} 1, & \left(\mathcal{C}'_{n+\|\mathcal{C}\|-m} \in hp(\mathcal{C}_{\|\mathcal{C}\|})\right) \wedge \\ & \qquad (\|\mathcal{C}'\| > \|\mathcal{C}\|+n-1-m) \\ 0, & \text{otherwise} \end{cases}$$

In summary, the total workload of the regular callback instances of $\mathcal{C}'^{j}$ executed in $[0, t_3]$ is upper-bounded by

$$e(\mathcal{C}'_{n+\|\mathcal{C}\|-m}) \cdot \varepsilon' + \sum_{z=1}^{\min(n+\|\mathcal{C}\|-1-m,\|\mathcal{C}'\|)} e(\mathcal{C}'_z) \qquad (12)$$

We can observe that (12) is non-increasing with respect to m. Therefore, (12) is maximized when m is set to its minimal possible value, i.e., $\mathcal{C}'^{j}_1$ executes in the earliest possible processing window. Since t_2 is in processing window pw_n and γ' is the number of instances of $\mathcal{C}'$ released in $[0, t_2]$, the earliest processing window in which $\mathcal{C}'^{\gamma'+1}_{tm}$ executes is pw_n, so the earliest processing window in which $\mathcal{C}'^{\gamma'+1}_1$ executes is pw_{n+1}. Then by Lemma 2, the earliest processing window in which $\mathcal{C}'^{j}_1$ executes is $pw_{n+j-\gamma'}$. Thus (12) is maximized when $m = n + j - \gamma'$, with which we can rewrite (12) as:

$$e(\mathcal{C}'_{n+\|\mathcal{C}\|-(n+j-\gamma')}) \cdot \varepsilon' + \sum_{z=1}^{\min(n+\|\mathcal{C}\|-1-(n+j-\gamma'),\|\mathcal{C}'\|)} e(\mathcal{C}'_z)$$

$$= e(\mathcal{C}'_{\mu'}) \cdot \varepsilon' + \sum_{z=1}^{\min(\mu'-1,\|\mathcal{C}'\|)} e(\mathcal{C}'_z) \quad \text{(substitute } \|\mathcal{C}\|-(j-\gamma') \text{ by } \mu')$$

Note that $\sum_{z=1}^{\min(\mu'-1,\|\mathcal{C}'\|)} = 0$ when $\mu' \leq 1$ and $e(\mathcal{C}'_{\mu'})\cdot\varepsilon' = 0$ when $\mu' < 1$, corresponding to the cases $m \geq n + \|\mathcal{C}\| - 1$ and $m > n + \|\mathcal{C}\| - 1$ when analyzing the workload executed before and in $pw_{n+\|\mathcal{C}\|-1}$, respectively.

So far we have proved that the workload of $\mathcal{C}'^{j}$ executed in $[t_1, t_3]$ is upper-bounded by

$$e(\mathcal{C}'_{tm}) + e(\mathcal{C}'_{\mu'}) \cdot \varepsilon' + \sum_{z=1}^{\min(\mu'-1,\|\mathcal{C}'\|)} e(\mathcal{C}'_z) \qquad (13)$$

Since the number of instances of $\mathcal{C}'$ released during $[0, t_3]$ is no larger than $\alpha_{\mathcal{C}'}(t_3)$, summing up the upper bound in (13) for each $\mathcal{C}'^{j}$ with $j \in [\gamma'+1, \alpha_{\mathcal{C}'}(t_3)]$ proves that the workload of part (ii) is upper-bounded by $\mathcal{M}'(\gamma', t_3)$. □

Acknowledgement

This work is supported by the Research Grants Council of Hong Kong (GRF 15204917 and 15213818) and National Natural Science Foundation of China (Grant No. 61672140).

References

[1] ROS-Introduction, "http://wiki.ros.org/ROS/Introduction."
[2] Robots using ROS, "http://robots.ros.org."
[3] Tully Foote, "ROS community metrics report," 2018.
[4] B. Giorgio, "Real-time issues in advanced robotics applications." in *8th Euromicro Workshop on Real-Time Systems*, 1996.
[5] ROS2 Overview, "https://index.ros.org/doc/ros2/."
[6] D. Casini, T. Blaß, I. Lütkebohle, and B. B. Brandenburg, "Response-time analysis of ros 2 processing chains under reservation-based scheduling," in *31st Euromicro Conference on Real-Time Systems (ECRTS)*, 2019.
[7] A. Corsaro, G. Pardo-Castellote, and C. Tucker, "Dds interoperability demo." Object Management Group, 2009.
[8] Real-Time Executor in micro-ROS, "https://micro-ros.github.io/docs/concepts/client_library/real-time_executor/."
[9] N. Valigi, "https://roscon.ros.org/2019/talks/roscon2019_concurrency.pdf," ROSCon 2019.
[10] Y. Saito, T. Azumi, S. Kato, and N. Nishio, "Priority and synchronization support for ROS." in *4th International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, 2016.
[11] W. Hongxing, 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 Generation Computer Systems*, vol. 56, pp. 171–178, 2016.
[12] Y. Saito, F. Sato, T. Azumi, S. Kato, and N. Nishio, "ROSCH:real-time scheduling framework for ROS," in *24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018.
[13] Y. Suzuki, T. Azumi, S. Kato, and N. Nishio, "Real-time ROS extension on transparent CPU/GPU coordination mechanism," in *21st International Symposium on Real-Time Distributed Computing (ISORC)*, 2018.
[14] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *13th International Conference on Embedded Software*, 2016.
[15] C. Gutierrez, L. Juan, I. Ugarte, and V. Vilches, "Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications." Technical report, Erle Robotics S.L., 2018.

[16] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, “End-to-end timing analysis of cause-effect chains in automotive embedded systems,” *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.
[17] J. Schlatow and R. Ernst, “Response-time analysis for task chains in communicating threads,” in *22nd Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2016.
[18] ——, “Response-time analysis for task chains with complex precedence and blocking relations,” *ACM Transactions on Embedded Computing Systems*, vol. 16, no. 5s, pp. 1–19, 2017.
[19] A. Girault, C. Prvot, 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*, vol. 37, no. 11, pp. 2578–2589, 2018.
[20] Z. Hammadeh, R. Ernst, S. Quinton, R. Henia, and L. Rioux, “Bounding deadline misses in weakly-hard real-time systems with task dependencies,” in *20th Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2017.
[21] S. Schliecker and R. Ernst, “A recursive approach to end-to-end path latency computation in heterogeneous multiprocessor systems,” in *7th IEEE/ACM International Conference on Hardware/software Codesign and System Synthesis (CODES+ISSS)*, 2009.
[22] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson, “A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics,” in *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems(CRTS)*, 2008.
[23] J. Martinez, I. S. anudo, and M. Bertogna, “Analytical characterization of end-to-end communication delays with logical execution time,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2244–2254, 2018.
[24] J. Abdullah, G. Dai, and W. Yi, “Worst-case cause-effect reaction latency in systems with non-blocking communication,” in *22nd Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019.
[25] M. Dürr, G. V. D. Brüggen, K. Chen, and J. Chen, “End-to-end timing analysis of sporadic cause-effect chains in distributed systems,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, p. 58, 2019.
[26] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, “System level performance analysis - the SymTA/S approach,” *IEEE Proceedings - Computers and Digital Techniques*, vol. 152, no. 2, pp. 148 – 166, 2005.
[27] K. Richter, “Compositional scheduling analysis using standard event models: The SymTA/S approach,” *PhD thesis*, 2005.
[28] M. Jersak, “Compositional performance analysis for complex embedded applications,” *PhD thesis*, 2005.
[29] K. Richter, M. Jersak, and R. Ernst, “A formal approach to MpSoC performance verification,” *Computer*, vol. 36, no. 4, pp. 60–67, 2003.
[30] J. L. Boudec and P. Thiran, “Network calculus - a theory of deterministic queuing systems for the internet,” in *LNCS 2050*. Springer Verlag, 2001.
[31] M. Joseph and P. Pandya, “Finding response times in a real-time system,” *The Computer Journal*, 1986.
[32] E. Wandeler and L. Thiele, “Real-Time Calculus (RTC) Toolbox,” 2006. [Online]. Available: http://www.mpa.ethz.ch/Rtctoolbox