8 Appendix

8.1 The background of ROS2 system

Fig.7 shows the architecture of ROS2. The ROS2 architecture can be divided into three layers: Application, Middleware and OS. As a middleware, ROS2 bridges the OS and Application Layers.

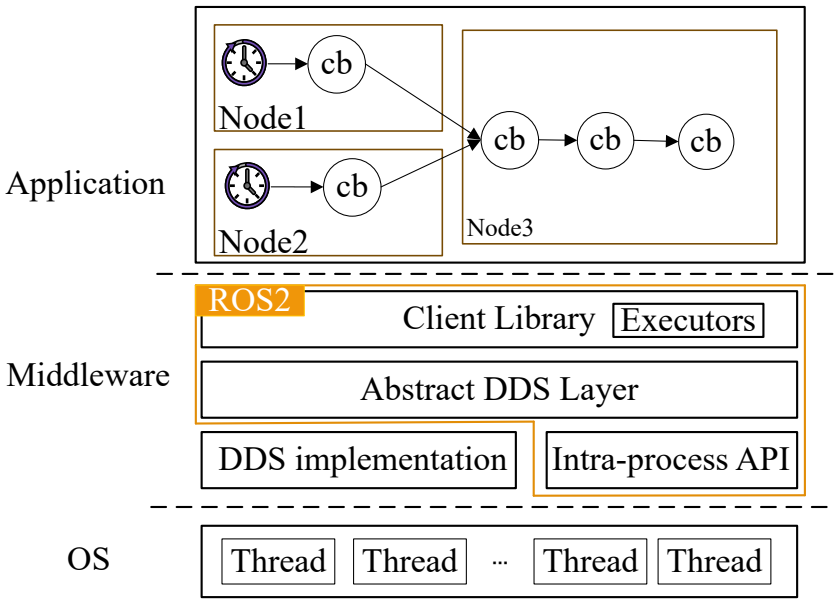

Fig. 7: ROS2 architecture.

ROS2 has five types of callbacks: timer, subscription, service, client, and waitable callbacks, which are the most basic units of execution. Timer callbacks are periodically triggered by the system's timer. The others are triggered by the message from external event. In the publish-subscribe mechanism, callbacks can publish messages to specific topics, triggering the callbacks that subscribe to those topics.

Node is a set of callbacks, with each node typically representing a specific module. Several nodes collaborate to simplify the code structure and enhance reusability. Nodes act as the minimum allocation unit to executors. With a node, all callbacks are executed by a single executor and cannot be distributed across multiple executors.

ROS2 applications are typically composed of a series of nodes. To deploy ROS2 applications, nodes must be allocated to executors which manage operating system threads to execute callbacks for different nodes.

Based on the scheduling abstraction of ROS2, processing chains (chains) are semantic abstractions defined by message exchanges between callbacks among

one or more nodes. ROS2 does not specify any properties of chains, nor do executors take into account the temporal and resource requirements of chains when scheduling callbacks. However, the response time of chains markedly influences the performance of real-time systems.

ROS2 multi-threaded executors use callback groups to control the parallel execution of callbacks. There are two types of callback groups: reentrant and mutually-exclusive. At runtime, the execution eligibility of instances released by callback c depends on the state of the callback group it belongs to:

- c belongs to a mutually-exclusive callback group, the released instances of c can be scheduled for execution by executors if there are no callback instances currently being executed in the callback group to which c belongs; Otherwise, the released instances of c will be blocked.
- c belongs to a reentrant callback group, instances of c can always be scheduled for execution by executors after being released.

8.2 The proof of Lemma 1 and Lemma 2

Lemma 1. *In ROS2 multi-threaded executors, for any chain Γ_i, there can be a maximum of $\lceil \frac{D_i}{T_i} \rceil$ schedulable instances simultaneously in the system.*

Proof. At time t, the Γ_i releases instance c, so the latest time for c to complete execution is $t + D_i$. During the next release period $t + T_i$ to $t + D_i$, Γ_i can release at most $\lceil \frac{D_i - T_i}{T_i} \rceil$ instances, and the latest completion time for these instances must be more than $t + D_i$. Therefore, for Γ_i, there can be a maximum of $\lceil \frac{D_i}{T_i} \rceil$ ($\lceil \frac{D_i - T_i}{T_i} \rceil + 1 = \lceil \frac{D_i}{T_i} \rceil$) schedulable instances simultaneously in the system.

Lemma 2. *In ROS2 multi-threaded executors, for any given chain Γ_i, the execution progress varies among instances coexisting simultaneously.*

Proof. In ROS2 multi-threaded executors, all threads maintain a common set, ***ReadySet***, which keeps track of the callback instances that are ready and waiting for execution. *ReadySet* is safeguarded by a mutually-exclusive lock, necessitating each thread to acquire the lock before scheduling callback instances. This means that only one thread can schedule callback instances at any moment. Furthermore, at any given time [4], the *ReadySet* is constrained to contain no more than a single instance of each callback. Hence, for any given chain Γ_i, the execution progress varies among instances coexisting simultaneously.

8.3 The proof of Theorem 1-3.

For the dbf (Δ) of ROS2 multi-threaded executors, it is essential to account the workload induced by the precedence dependencies between callbacks in chains. [11] derived the following Lemma:

- (Lemma 3 in [11]) Consider two adjacent callbacks of Γ_i, $C_{\langle i,j \rangle}$ and $C_{\langle i,j+1 \rangle}$, on a multithreaded executor. The precedence dependency blocking caused by $C_{\langle i,j \rangle}$ introduces an additional workload as interference to the start of $C_{\langle i,j+1 \rangle}$, which is upper-bounded by $m \cdot E_{\langle i,j \rangle}$.

– (Lemma 4 in [11]) For the last callback of Γ_i, the precedence dependency blocking caused by all of its preceding callbacks is given by $m \cdot \big(E_i - E_{\langle i,\|\Gamma_i\|-1\rangle}\big)$.

Theorem 1. *On a ROS2 multi-threaded executor with m threads, based on the CIL-EDF scheduling scheme, if $dbf(\Delta) < sbf_\omega(\Delta)$, the response time of chain $\Gamma_i = \big\{C_{<i,0>}, C_{<i,1>}, \ldots, C_{<i,\|\Gamma_i\|-1>}\big\}$ with a constrained deadline can be upper-bounded by $R_i = \Delta + \overline{sbf}_c(E_{<i,||\Gamma_i||-1>} - 1)$, holds for the following $dbf(\Delta)$:*

$$\begin{aligned} dbf(\Delta) =& m \cdot (E_i - E_{\langle i,\|\Gamma_i\|-1\rangle}) \\ &+ \sum_{\forall \Gamma_j \in \Gamma \wedge D_j < D_i} \left\lfloor \frac{\Delta}{T_j} \right\rfloor \cdot E_j + \min(E_j, \Delta - \left\lfloor \frac{\Delta}{T_j} \right\rfloor \cdot T_j) \\ &+ \sum_{\forall \Gamma_k \in \Gamma - \{\Gamma_i\}} \min(E_k, \Delta) \end{aligned} \tag{6}$$

$dbf(\Delta)$ can be solved through fixed-point iteration with an initial condition of $\Delta = E_i - E_{\langle i,\|\Gamma_i\|-1\rangle}$.

Proof. Let t be the release time of an instance of Γ_i. On a ROS2 multi-threaded executor based on the CIL-EDF scheduling scheme, $dbf\,(\Delta)$ for Γ_i with a constrained deadline requires upper-bound the following workload: (i) the workload caused by precedence dependencies. (ii) the workload caused by interference from high-priority chain instances released within the $(t, t + \Delta)$. (iii) the workload caused by interference or blocking by chain instances released at or before t.

For (i), by Lemma 4 in [13], the maximum workload caused by precedence dependencies is $m \cdot \big(E_i - E_{\langle i,\|\Gamma_i\|-1\rangle}\big)$.

For (ii), Let t_j be the release time of an interfering chain Γ_j instance, where $t < t_j < t + \Delta$. The proof proceeds by contradiction. Suppose there exists an instance of Γ_j with a priority higher than that of Γ_i instance, and $D_j \geq D_i$. Since the executor based on the CIL-EDF scheduling scheme, $t + D_i > t_j + D_j$ holds. However, given the fact $t < t_j$, it means $t + D_i > t_j + D_j$ cannot be true. Thus, the assumption is incorrect. Therefore, if the priority of the Γ_j instance released within the $(t, t + \Delta)$ is higher than that of the Γ_i instance, it must be satisfied $D_j < D_i$. Within the $(t, t + \Delta)$, the Γ_j can release at most $\lfloor \frac{\Delta}{T_j} \rfloor + 1$ instances. The last instance released may not finish execution within this interval, the workload caused by this last instance can be upper-bounded by $\min\Big(E_j, \Delta - \lfloor \frac{\Delta}{T_j} \rfloor \cdot T_j\Big)$. In conclusion, the workload caused by interference from high-priority chain instances released within the $(t, t + \Delta)$ can be upper-bounded by $\sum_{\forall \Gamma_j \in \Gamma \wedge D_j < D_i} \left\lfloor \frac{\Delta}{T_j} \right\rfloor \cdot E_j + \min(E_j, \Delta - \left\lfloor \frac{\Delta}{T_j} \right\rfloor \cdot T_j)$.

For (iii), Since the Γ_i with a constrained deadline, instances do not suffer from selfinterference. Let the release time of interfering chain Γ_k instances be t_k, where $t_k < t$. Regardless of D_i or D_k being larger, $t + D_i$ may be larger $t_k + D_k$, indicating a higher priority for Γ_k instances over Γ_k instances. Therefore, all other chains should be considered as interfering chains, have $\forall \Gamma_k \in \Gamma - \{\Gamma_i\}$. At $t + \Delta$, the last callback instance of the Γ_i starts execution and won't be

preempted, limiting the workload caused by Γ_k instances to Δ, meaning the workload can be upper-bounded by $\min(E_k, \Delta)$. In conclusion, the workload caused by interference or blocking by chain instances released at or before t can be upper-bounded by $\sum_{\forall \Gamma_k \in \Gamma - \{\Gamma_i\}} \min(E_k, \Delta)$.

Theorem 2. *On a ROS2 multi-threaded executor with m threads, based on the CIL-EDF scheduling scheme, if $dbf(\Delta) < sbf_\omega(\Delta)$, the response time of chain $\Gamma_i = \{C_{<i,0>}, C_{<i,1>}, ..., C_{<i,\|\Gamma_i\|-1>}\}$ with an arbitrary deadline can be upper-bounded by $R_i = \Delta + \overline{sbf_c}(E_{<i,||\Gamma_i||-1>} - 1)$, holds for the following $dbf(\Delta)$:*

$$\begin{aligned} dbf(\Delta) =& m \cdot (E_i - E_{\langle i,\|\Gamma_i\|-1\rangle}) \\ &+ \sum_{\forall \Gamma_j \in \Gamma \wedge D_j < D_i} \left\lceil \frac{\Delta}{T_j} \right\rceil \cdot E_j \\ &+ (\sum_{\forall \Gamma_k \in \Gamma} \left\lceil \frac{D_k}{T_k} \right\rceil \cdot \min(E_k, \Delta) - \min(E_i, \Delta)) \end{aligned} \tag{7}$$

$dbf(\Delta)$ can be solved through fixed-point iteration with an initial condition of $\Delta = E_i - E_{\langle i,\|\Gamma_i\|-1\rangle}$.

Proof. For the $dbf(\Delta)$ of Γ_i with an arbitrary deadline, similar to Theorem 1, consist of three parts. The distinctions lie in (ii) and (iii) of Theorem 2.

For (ii), it's possible for the next instance of a chain to start execution while the preceding instance hasn't finished, as chains with an arbitrary deadline may have release periods smaller than relative deadlines. Therefore, for the interfering chain Γ_j, the maximum workload of instances released within the interval Δ is $\lceil \frac{\Delta}{T_j} \rceil \cdot E_j$. Similar to (ii) in Theorem 1, if Γ_j instances released within the $(t, t + \Delta)$ have higher priority than Γ_i instances, it must be satisfied $D_j < D_i$. In conclusion, the workload caused by interference from high-priority chain instances released within the $\sum_{\forall \Gamma_j \in \Gamma \wedge D_j < D_i} \left\lceil \frac{\Delta}{T_j} \right\rceil \cdot E_j$.

For (iii), Since Γ_i is a chain with an arbitrary deadline, it may have multiple instances simultaneously. Therefore, instances may suffer from self-interference. Thus, have $\forall \Gamma_k \in \Gamma$. For Γ_k, by Lemma 1, the number of simultaneous instances can be upper-bounded by $\lceil \frac{D_k}{T_k} \rceil$. For Γ_i, $\lceil \frac{D_i}{T_i} \rceil$ including the one being analyzed. To avoid redundancy, we need to subtract $\min(E_i, \Delta)$. In conclusion, the workload caused by interference or blocking by chain instances released at or before t can be upper-bounded by $(\sum_{\forall \Gamma_k \in \Gamma} \left\lceil \frac{D_k}{T_k} \right\rceil \cdot \min(E_k, \Delta) - \min(E_i, \Delta))$.

Theorem 3. *With mutually-exclusive callback groups, On a ROS2 multi-threaded executor with m threads, based on the CIL-EDF scheduling scheme, if $dbf(\Delta) < sbf_\omega(\Delta)$, the response time of the chain $\Gamma_i = \{C_{<i,0>}, C_{<i,1>}, ..., C_{<i,\|\Gamma_i\|-1>}\}$ can be upper-bounded by $R_i = \Delta + \overline{sbf_c}(E_{<i,||\Gamma_i||-1>} - 1)$, holds for the following $dbf(\Delta)$:*

$$\begin{aligned} dbf(\Delta) =& RHS\ of\ Eq.(6)\ or\ Eq.(7) \\ &+ \textstyle\sum_{j=0}^{\|\Gamma_i\|-1} m \cdot mutually\_exclusive\_blocking\_time(C_{<i,j>}, \Delta) \end{aligned} \tag{8}$$

Algorithm 2 *mutually_exclusive_blocking_time* $(C_{<i,j>}, \Delta)$

1: $blocking\_time = 0$
2: $group = callback\_group(C_{<i,j>}) - \{C_{<i,j>}\}$
3: ***for all*** $C_{<k,y>} \in group$ ***do***
4: ***if*** $k \neq i$ ***then***
5: $blocking\_time+ = \left\lceil \frac{D_k}{T_k} \right\rceil \cdot E_{<k,y>}$
6: ***if*** $D_k < D_i$ ***then***
7: $blocking\_time+ = \left\lceil \frac{\Delta}{T_k} \right\rceil \cdot E_{<k,y>}$
8: ***end if***
9: ***end if***
10: ***end for***
11: ***return*** $blocking\_time$

Proof. Theorem 3, extending Theorem 1 and Theorem 2, incorporates the workload caused by mutually-exclusive callback groups to the end of $dbf\,(\Delta)$.

Alg.1 provides an upper-bound of the workload that $C_{<i,j>}$ is blocked within the $(t, t+\Delta)$ due to the effect of mutually-exclusive callback groups. Since the blocking time from callbacks of the same chain has been already accounted as precedence-dependency blocking in (i) of Eq.(5) or Eq.(6), Alg.1 excludes the blocking from group-mate callbacks that are from the Γ_i (line 4 of Alg.1). $C_{<i,j>}$ instance can be blocked by multiple instances of its higher-priority groupmates, it consists of two parts: (i) Γ_k instances released at or before t. by Lemma 1, the number of simultaneous instances can be upper-bounded by $\lceil \frac{D_k}{T_k} \rceil$. Therefore, the number of $C_{<k,y>}$ instances with higher priority than $C_{<i,j>}$ instances can be upper-bounded by $\lceil \frac{D_k}{T_k} \rceil$ (line 5 of Alg.1). (ii) Γ_k instances released within the $(t, t+\Delta)$. If the $C_{<k,y>}$ instance with higher-priority than the $C_{<i,j>}$ instance, it must satisfy $D_k < D_i$ (line 6 of Alg.1). In the time interval Δ, for the Γ_k, the number of released instances can be upper-bounded by $\lceil \frac{\Delta}{T_k} \rceil$, thus, for the $C_{<k,y>}$, the number of released instances can be upper-bounded by $\lceil \frac{\Delta}{T_k} \rceil$ (line 7 of Alg.1).

By Lemma 3 in [13], the workload caused by the effect of mutually-exclusive callback groups on $C_{<i,j>}$ can be upper-bounded by $m \cdot mutually\_execlusive\_blocking\_time(C_{<i,j>}, \Delta)$. When all callbacks of the Γ_i are in different mutually-exclusive callback groups, and all of their group-mates block their execution as long as possible, the effect of mutually-exclusive callback groups for the Γ_i is maximized. Therefore, we need to account the workload caused by all callbacks of the Γ_i due to mutually-exclusive callback groups as part of $dbf\,(\Delta)$.

8.4 The detailed configuration of evaluation.

We use the Foxy version of ROS2 on an Ubuntu 20.04 LTS with kernel 5.4.0-26-generic and a desktop with Intel i7-1260P@2495.994 MHZ and a total memory of 16 GB (four cores are used in our experiments). Each thread of multi-threaded executors is set to the SCHED_FIFO Linux scheduling class.Table 1 and Table

2 present the specifications of the chain set for case studies, which is derived from the configuration and workload of tasks executed by autonomous robots. In the chain set, the first callback of the Γ_0 and Γ_1 share a common callback. Evaluations for chains with arbitrary deadlines have twice the workload than chains with constrained deadlines.

Table 1: Specifications for chains with constrained deadlines set.

Chains	Specifications(msec)
$\Gamma_0 = \{C_{<0,0>}, C_{<0,1>}\}$	$E_{<0,0>} = 2, E_{<0,1>} = 16, T_0 = 80, D_0 = 80$
$\Gamma_1 = \{C_{<0,0>}, C_{<1,1>}, C_{<1,2>}, C_{<1,3>}\}$	$E_{<0,0>} = 2, E_{<1,1..3>} = \{2, 9, 9\}, T_1 = 80, D_1 = 80$
$\Gamma_2 = \{C_{<2,0>}, C_{<2,1>}, C_{<2,2>}, C_{<2,3>}\}$	$E_{<2,0..3>} = \{21, 8, 7, 2\}, T_2 = 120, D_2 = 120$
$\Gamma_3 = \{C_{<3,0>}, C_{<3,1>}, C_{<3,2>}\}$	$E_{<3,0..3>} = \{23, 8, 14\}, T_3 = 140, D_3 = 140$
$\Gamma_4 = \{C_{<4,0>}, C_{<4,1>}, C_{<4,2>}, C_{<4,3>}\}$	$E_{<4,0..3>} = \{18, 11, 8, 8\}, T_4 = 160, D_4 = 160$
$\Gamma_5 = \{C_{<5,0>}, C_{<5,1>}, C_{<5,2>}, C_{<5,3>}\}$	$E_{<5,0..3>} = \{11, 22, 5, 18\}, T_5 = 180, D_5 = 180$

Table 2: Specifications for chains with arbitrary deadlines set.

Chains	Specifications(msec)
$\Gamma_0 = \{C_{<0,0>}, C_{<0,1>}\}$	$E_{<0,0>} = 2, E_{<0,1>} = 16, T_0 = 40, D_0 = 80$
$\Gamma_1 = \{C_{<0,0>}, C_{<1,1>}, C_{<1,2>}, C_{<1,3>}\}$	$E_{<0,0>} = 2, E_{<1,1..3>} = \{2, 9, 9\}, T_1 = 40, D_1 = 80$
$\Gamma_2 = \{C_{<2,0>}, C_{<2,1>}, C_{<2,2>}, C_{<2,3>}\}$	$E_{<2,0..3>} = \{21, 8, 7, 2\}, T_2 = 60, D_2 = 120$
$\Gamma_3 = \{C_{<3,0>}, C_{<3,1>}, C_{<3,2>}\}$	$E_{<3,0..3>} = \{23, 8, 14\}, T_3 = 70, D_3 = 140$
$\Gamma_4 = \{C_{<4,0>}, C_{<4,1>}, C_{<4,2>}, C_{<4,3>}\}$	$E_{<4,0..3>} = \{18, 11, 8, 8\}, T_4 = 80, D_4 = 160$
$\Gamma_5 = \{C_{<5,0>}, C_{<5,1>}, C_{<5,2>}, C_{<5,3>}\}$	$E_{<5,0..3>} = \{11, 22, 5, 18\}, T_5 = 90, D_5 = 180$

8.5 The detailed analysis of schedulability.

Table 3 and Table 4 present the schedulability of each chain under different configurations in vector form ($[\Gamma_0, \Gamma_0, ..., \Gamma_5]$). In the evaluation of chains with constrained deadlines, all cases, the CIL-EDF scheduling scheme guarantees schedulability for all chains, but for the other two, will have a small number of chain instances that do not finish execut ing before their deadline. In the evaluation of chains with arbitrary deadlines, similar to the comparison of WCRT mentioned above, except for the schedulability of Γ_0 being better under the PiCAS scheme compared to the CIL-EDF scheduling scheme, in most cases, the schedulability of chains under CIL-EDF scheduling scheme is significantly better than the other two.

Table 3: compare the schedulability of chains with constrained deadlines

	2	3
Default	[0.998,0.945,0.99,0.997,0.994,1]	[1, 0.998, 0.997, 1, 1, 1]
PiCAS	[1, 0.998, 1, 1, 0.997, 0.97]	[1,0.995,0.997,1,0.996,0.996]
CIL-EDF	**[1, 1, 1, 1, 1, 1]**	**[1, 1, 1, 1, 1, 1]**

(a) CD-R2~3

	2	3
Default	[1, 0.917, 0.998, 1, 0.990, 1]	[1, 0.998, 1, 1, 1, 1]
PiCAS	[1, 1, 1, 1, 0.443, 0.975]	[1, 0.998, 1, 1, 0.997, 1]
CIL-EDF	**[1, 1, 1, 1, 1, 1]**	**[1, 1, 1, 1, 1, 1]**

(b) CD-M2~3

Table 4: compare the schedulability of chains with arbitrary deadlines

	2	3	4
Default	[0, 0, 0, 0, 0, 0]	[0.009,0,0.001,0.010,0.002,0]	[0.717,0.718,0.714,0.715,0.714, 0.714]
PiCAS	[1, 0.011, 0, **0**, **0**]	[0.994,0.153,0.179,0.460,0.002,0]	[1, 1, 1, 1, 0.900, 0.781]
CIL-EDF	**[0.480,0.047,0.185,0.058,0.122,0.097]**	**[0.725,0.587,0.593,0.856,0.810,0.677]**	**[1, 1, 1, 1, 1, 1]**

(a) AD- R2~4

	2	3	4
Default	[0.002, 0, 0, 0, 0, 0]	[0.032,0.002,0.009,0.008,0, 0.982]	[0.998, 0.375, 1, 1, 0.170, 1]
PiCAS	[0.652, 0.025, 0, **0**, **0**]	[1, 0.408, 0.402, 0.054, 0, 0]	[1, 0.990, 1, 1, 0.065, 0.925]
CIL-EDF	**[0.676,0.086,0.313,0.316,0.340,0.091]**	**[0.998,0.992,0.996,0.974,0.939, 0.902]**	**[1, 1, 1, 1, 1, 1]**

(b)AD- M2~4