Worst-Case Latency Analysis of Message Synchronization in ROS

Ruoxiang Li[1,2], Xu Jiang[3], Zheng Dong[4], Jen-Ming Wu[5], Chun Jason Xue[1] and Nan Guan[1]

[1]City University of Hong Kong, Hong Kong SAR
[2]City University of Hong Kong Shenzhen Futian Research Institute, China
[3]Northeastern University, China [4]Wayne State University, US
[5]Hon Hai Research Institute, Taiwan

***Abstract*—Multi-sensor data fusion is crucial for modern autonomous systems to accurately perceive their surrounding environments and make intelligent decisions. However, as different sensor sources may have significant time disparity, it is necessary to synchronize their data before sending them to the fusion algorithm, in order to control such differences and get meaningful fusion results. This paper discusses the message synchronization policy in ROS, a popular framework for robotic systems. The ROS message synchronization policy has proven to be highly effective in reducing the time disparity, but it introduces a certain level of latency. Therefore, to use it for real-time systems, it is essential to establish an upper bound for the worst-case latency that may occur. Specifically, we analyze two key latency metrics of the ROS message synchronization policy, the *passing latency* and *reaction latency*, which are needed to analyze the end-to-end delay and reaction time on the system level. We conduct experiments under different settings to evaluate the precision of our proposed latency upper bounds against the maximum observed latency in real execution.**

I. Introduction

Multi-sensor data fusion plays a crucial role in the operation of modern autonomous systems, including autonomous vehicles, robots, and drones. By using this technology, these systems are empowered to perceive and interpret their surrounding physical environment accurately. This capability enables them to make intelligent decisions and execute complex tasks with precision and reliability. However, in practice, the sensor data obtained from different sources may not have perfectly aligned sampling time, and they may experience varying delays before reaching the fusion algorithm [1], [2]. Consequently, the input data from these diverse sensor sources can exhibit significant *time disparity*, which indicates the time difference between their actual sampling instances [3]. If the time disparity is substantial, the fusion results may lose their usefulness or even become completely meaningless. Therefore, it is important to synchronize the data from different sensor sources before transmitting them to the fusion algorithm to manage the time disparity.

In this paper, we examine the message synchronization policy in ROS (Robotic Operating System) [4]–[6], a widely used software framework for developing robotic systems. With its extensive adoption by countless developers, ROS has powered a diverse range of robots and autonomous systems. Recent research [3], [7] has demonstrated excellent performance of ROS's message synchronization policy in terms of minimizing time disparity and outperforms its competitors, such as the one employed by Apollo Cyber RT [8]. Especially, The message synchronization policy used in ROS-based applications, such as Autoware [9], has proven effective and could be adopted in other systems.

While the ROS message synchronization policy effectively mitigates time disparity, it introduces a trade-off in the form of latency. This latency arises from the temporary buffering of messages until they can be grouped with others that share similar sampling times, which is the key to reducing the overall time disparity. The incurred latency plays a significant role in affecting the timing behavior of ROS applications. To ensure the applicability of the ROS message synchronization policy in real-time systems, it is crucial to quantify the extent of this latency and, more importantly, establish a safe upper bound for its worst-case scenario. However, the challenge lies in the absence of a definitive solution for bounding the worst-case latency resulting from the ROS message synchronization policy. Addressing this issue is the goal of this paper.

In this study, we examine two types of latency metrics associated with the ROS message synchronization policy, namely, the *passing latency* and the *reaction latency*. The passing latency refers to the time gap between the time when a message arrives and when it leaves the message synchronizer. This metric is useful in determining the *end-to-end delay* for sensor data going through the processing pipeline. Reaction latency, on the other hand, considers both the passing latency of a message and the time delay resulting from discarded messages preceding it. This metric is crucial in calculating another important system-level real-time performance metric, i.e., *end-to-end reaction time* [10]–[14]. Section II will provide a detailed explanation of the passing and reaction latency and how they relate to end-to-end delay and reaction time.

We conduct experiments under different settings, including the different number of channels, the varied data sampling periods, and the random delay time experienced by messages before arriving at the synchronization policy, to evaluate the precision of our proposed latency upper bounds against the maximum observed latency in real execution.

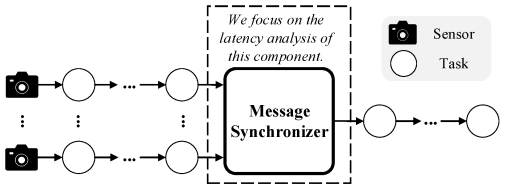


Fig. 1: A system example with a message synchronizer.

II. Problem Definition

A. System Model

The ROS message synchronizer, called *synchronizer* for short, is a software component to synchronize messages from different sensor sources before sending them to the data fusion component (as shown in Fig. 1). The synchronizer has N input channels and 1 output channel. Each input channel of the synchronizer has a buffer queue Q_i to temporally store messages arrived at this channel. We assume that each queue Q_i is sufficiently long. This assumption does not pose any limitation to our study since there is a known upper bound of the size of each queue in ROS message synchronizer, by which we can easily set a proper queue size to meet this assumption [3]. For simplicity of presentation, we also use Q_i to refer to the i-th input channel of the Message Synchronizer, when it is unambiguous from the context. The synchronizer selects one message from each input channel and combines them into an *output message set* according to some policy, which is released to the output channel. We will introduce the policy used by the ROS message synchronizer in Section III.

We use m_i^k to represent the k-th message currently in queue Q_i. Also, we use $m_i^{\rho(i)}$[1] to denote the $\rho(i)$-th message among all the messages coming from the i-th input channel. It is worth noting that the difference between the notations m_i^k and $m_i^{\rho(i)}$ is that m_i^k only represents the message that currently in Q_i, while $m_i^{\rho(i)}$ denotes the message currently in Q_i or the one that has already been discarded from Q_i, which considers the total number of messages from the i-th input channel. For simplicity, sometimes we also use m_i to represent a message in Q_i when there is no need to specify which message it is exactly. Each message m_i^k has two key timing characteristics:

- $\tau(m_i^k)$, the *timestamp* of m_i^k, which is the time point when the sensor data carried by m_i^k was sampled[2].
- $\alpha(m_i^k)$, the *arrival time* of m_i^k, which is the time point when the m_i^k arrives at the synchronizer.

In general, $\tau(m_i^k)$ is smaller than $\alpha(m_i^k)$, since a message may experience some delay before arriving at the synchronizer due to, e.g., processing or transmission. We also assume that the messages in each channel arrive at the synchronizer in the same order as their timestamps.

We do *not* assume to know the exact $\tau(m_i^k)$ and $\alpha(m_i^k)$ of each message m_i^k. Instead, we assume to know the following timing parameters for messages in each channel Q_i:

- The minimum and maximum time difference between the timestamps of two consecutive messages in Q_i, denoted by T_i^B and T_i^W. We have $\forall k$:
$$T_i^B \leq \tau(m_i^{k+1}) - \tau(m_i^k) \leq T_i^W$$
- The minimum and maximum delay experienced by each message in Q_i, denoted by D_i^B and D_i^W. We have $\forall k$:
$$D_i^B \leq \alpha(m_i^k) - \tau(m_i^k) \leq D_i^W$$

The value of T_i^B and T_i^W are decided by the corresponding sensor's sampling rate. In some cases, the sensor data are sampled in a strictly periodic manner, which is a special case of our model where $T_i^B = T_i^W$. However, in reality, sensor data sampling is typically not perfectly periodic and can suffer from jitters due to numerous reasons such as software and hardware limitations [15]. Therefore, in this work, we presume the general case in which T_i^B and T_i^W may be different.

D_i^B and D_i^W refer to the processing and transmission delays experienced by a message before it reaches the synchronizer. For instance, a message is processed by a processing task (such as a *callback* in ROS) before it arrives at the synchronizer, then D_i^B (D_i^W) represents the best-case (worst-case) response time of this processing task plus the best-case (worst-case) transmission delay, i.e., the time between the completion of this task and the arrival of the corresponding message at the synchronizer. Response time analysis for real-time tasks is well-studied in existing real-time scheduling theory. There are many mature techniques to bound the best/worst-case response times under many different settings, including those based on the ROS executor and its variance [16]–[24]. And, transmission delay can also be theoretically analyzed [25]–[27] or pragmatically measured. Therefore, in this paper, we will not further explore how to estimate D_i^B and D_i^W in various system settings but focus on the new challenge we face in this work, i.e., analyzing the latency incurred by the synchronizer given known D_i^B and D_i^W estimations.

B. Passing Latency and Reaction Latency

We aim to analyze two types of latency metrics for the synchronizer, the passing latency and the reaction latency.

Definition 1 (Passing Latency). *If a message $m_i^{\rho(i)}$ is selected into an output message set published at time t_f, the passing latency of $m_i^{\rho(i)}$ is defined as $t_f - \alpha(m_i^{\rho(i)})$.*

In the synchronizer, not all received messages can be selected into the output message sets. Before a message is selected into an output message set, there is a possibility of discarding certain messages.

Definition 2 (Reaction Latency). *If a message $m_i^{\rho(i)}$ is selected into an output message set published at time t_f, and*

[1] ρ can be a function whose specific form is not important in our context, as we do not require the specification of which message $m_i^{\rho(i)}$ is.

[2] A message may pass through several processing tasks (callbacks in ROS) before reaching the synchronizer. In reality, a callback "consumes" an input message and "produces" an output message that inherits the timestamp of the input message. To simplify our abstract model and focus on the problem under study, we omit details about callbacks consuming input messages and producing output messages, and instead consider that the initial message goes through all processing tasks and eventually reaches the synchronizer.

$m_i^{\rho(i)-x}$ *($x \in \mathbb{N}^+$) is the last message before* $m_i^{\rho(i)}$ *that was selected into an output message set, the reaction latency of* $m_i^{\rho(i)}$ *is defined as* $t_f - \alpha(m_i^{\rho(i)-x})$.

The *worst-case passing latency* of a message channel is the maximum passing latency of among messages in this channel that are selected into output message sets. Similarly, the *worst-case reaction latency* of a message channel is the maximum reaction latency among all messages in this channel that are selected into output message sets.

Briefly speaking, the difference between passing latency and reaction latency is that the reaction latency includes both the passing latency and the extra latency caused by the discarded messages. As a special case, if no messages have been discarded in a channel, then the passing latency and reaction latency are identical for messages in this channel.

The passing latency is useful to track the *end-to-end delay* for the information carried by a message to traverse the entire processing system. For example, suppose a message is selected into an output message set by the synchronizer, which is further sent to the downstream fusion task and eventually generates a control signal to the actuator. By tracking the end-to-end delay for the message leading to each control signal, we can assess the staleness of status information relied upon by the control signal. This information is crucial for designing proper measures to compensate for this staleness when generating the control command.

The reaction latency is useful to calculate the *end-to-end reaction time* of the system regarding a sensor source. Loosely speaking, the reaction time of a processing pipeline is the time for it to react to an external event. The reaction latency associated with the synchronizer is caused by discarding messages and waiting for messages from separate channels to synchronize. For example, suppose a system consisting of two sensors for data sampling. Suppose one of these sensors experiences an extended sampling period. In that case, the synchronizer will have to wait for the messages from this sensor to arrive, despite already having received messages from the other sensor. This metric is essential for measuring a system's reaction time to external events. We will use the following example to illustrate the passing latency and reaction latency, as well their relationships with the *end-to-end* delay and reaction time.

C. An Illustrative Example

We use Fig. 2 to illustrate the passing and reaction latency, as well as their relationship with the end-to-end delay and end-to-end reaction time. Fig. 2-(a) depicts a system comprising two tasks for sampling sensor data, a synchronizer, a data fusion task, and an actuator task. For ease of presentation, we suppose that each task, including the synchronizer, is executed on a dedicated processor without any interference, and there is no communication delay between any two tasks. Suppose that $T_1^B = T_1^W = 6$ and $T_2^B = T_2^W = 20$. Moreover, we set the worst-case execution time of sensor task 1, sensor task 2, data fusion task, and actuator task as 1, 4, 2, and 2,

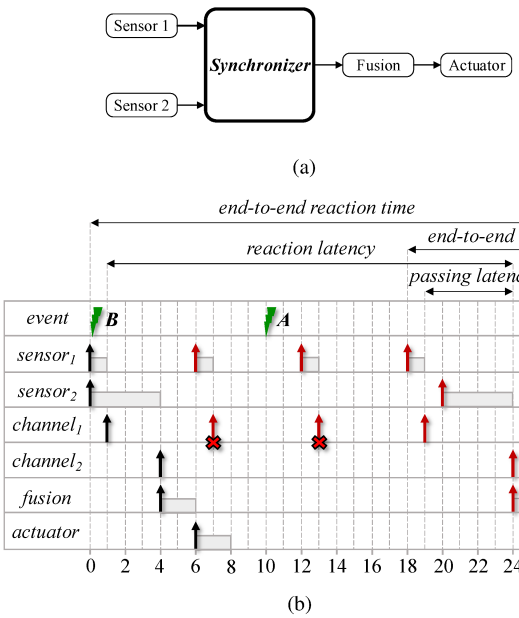


Fig. 2: Illustration of the passing and reaction latency. (a) A system includes two sensor data sampling tasks, a synchronizer, a data fusion task, and an actuator task. (b) An execution sequence example illustrating the relationship between the passing latency (and reaction latency) we focus on and the end-to-end delay (and the end-to-end reaction time). The dark upward arrows indicate the arrivals of messages sampled before the occurrence of event **B** that occurs right after time 0, and the red upward arrows denote the arrivals of messages that identify event **B**.

respectively. And the time required for the synchronizer to choose an output message set is negligible and has been set to 0. Accordingly, Fig. 2-(b) illustrates the execution sequence of the system shown in Fig. 2. Upon completion of execution of sensor task 1 or 2, the corresponding message is sent to the synchronizer. Fig. 2-(b), the synchronizer is represented by two channels to demonstrate the received messages from the sensor task 1 and 2. Once an output message set is selected, the synchronizer sends the message set (which comprises two messages, each from one of the channels) to the fusion task.

At time 1, a message sent by sensor task 1 arrives at the synchronizer. At time 4, the synchronizer receives a message sent by sensor task 2, and it combines this message with the one from channel 1 as an output message set, which is sent to the fusion task at time 4. After processing, the fusion task sends the message to the actuator task, which completes execution at time 8. Right after time 0, an external event **B** occurs, which is first captured by sensor 1 at time 6. And then, three messages, which were published by sensor task 1 identifying event **B**, arrive at the synchronizer at time 7, 13, and 19, respectively. As no new messages are received from channel 2, these three messages are buffered, awaiting messages from channel 2. At time 24, a new message sent by sensor task 2 arrives. At the same time, the synchronizer

combines it together with the message, which arrives at time 19 into an output message set, which is sent to the fusion task. Then, the fusion task sends the message to the actuator task at time 26, and the actuator task finishes its execution at time 28. Note that the messages arriving at time 7 and 13 will be discarded and not included in any output message sets.

In this example, the passing latency of the message arriving at time 19 caused by the synchronizer is the time difference between its arrival at time 19 and the publishing of the output message set at time 24, i.e., $24 - 19 = 5$. The corresponding end-to-end delay is the duration from the start time of sensor task 1 at time 18 to the completion of the actuator task at time 28, i.e., $28 - 18 = 10$, which includes the passing latency from time 19 to 24. The reaction latency is the time duration from the arrival of the message at time 1 to the publishing of the output message set at time 24, i.e., $24 - 1 = 23$, which includes the passing latency and the extra latency caused by the discarded messages. And the corresponding end-to-end reaction time is the time duration from the occurrence of event **B** at time 0 to the completion of the actuator task at time 28, i.e., $28 - 0 = 28$, which includes the reaction latency from time 1 to 24.

It is worth mentioning that the reaction latency of the synchronizer is defined regarding the arrival time of the last non-discarded message but not the occurrence time of the external event, which seems problematic. For example, if an external event **A** occurs at time 10, then the end-to-end reaction time regarding this event should be $28 - 10 = 18$. However, by our definition, the reaction latency of the synchronizer is $24 - 1 = 23$, which is larger than the end-to-end reaction time $28 - 10 = 18$. This is actually not a problem as our interest is to analyze the *worst-case* end-to-end reaction time no matter when the event actually occurs. The worst-case scenario is that the external event happens right after the sampling time of the last non-discarded message (event **B** in Fig. 2-(b)). Therefore, the worst-case time gap between the occurrence of event **B** and the generation of the first output message group containing the information of event **B** ($24 - 0$ in this example) equals the sum of two parts (1) the difference between the timestamp of the last non-discarded message and its arrival time to the synchronizer ($1 - 0$ in this example) and (2) the reaction latency ($24 - 1$ in this example). The former can be bounded using existing response time analysis techniques, while analyzing the latter is the goal of this paper. In summary, we define the reaction latency of the synchronizer assuming the worst-case scenario, i.e., the external event occurs right after the sampling time of the last non-discarded message. In this way, the definition of the reaction latency is simple yet sufficient to serve the purpose of bounding the worst-case end-to-end reaction time.

III. ROS Message Synchronization Policy

There are two synchronization policies in ROS, i.e., the *Exact Time* policy [28] and the *Approximate Time* policy [29]. The *Exact Time* policy only combines messages from different input channels with exactly the same timestamp into an output set and discards any messages without an exact match. As a result, any output message set published under the *Exact Time* policy will have a time disparity of 0. However, in reality, it is too restrictive to require data from different sensors to have exactly the same timestamp, so the *Exact Time* policy is rarely used in practice. Consequently, we focus on the *Approximate Time policy* in this paper, which is used to combine messages under a certain tolerance of time disparity. Please note that the model and results of this paper apply to both ROS 1 (the first generation of ROS) and ROS 2. More specifically, the *Approximate Time policy* is the same for all ROS 1 C++ versions since Diamondback and ROS 2 C++ versions until the latest Rolling, which was also stated in [3]. For the sake of brevity, we use the term "ROS" in this paper to include both ROS 1 and ROS 2. Throughout the remainder of this paper, we will use the term "policy" or "synchronization policy" interchangeably to represent the *Approximate Time* policy. In this paper, we adopt the abstract model presented in [3], but to keep our paper self-contained, we will provide a detailed explanation of this model in its entirety. We first define some concepts, followed by the abstract policy model.

We use $S = \{m_1, ..., m_N\}$ to denote a *regular set* containing N messages, each of which comes from a different queue. The time disparity of a regular set is defined as:

Definition 3 (Time Disparity)**.** *Let $S = \{m_1, ..., m_N\}$ be a regular set. The time disparity of S, denoted by $\Delta(S)$, is the maximum difference between the timestamps of the messages in S, i.e.,*

$$\Delta(S) = \max_{m_i \in S}\{\tau(m_i)\} - \min_{m_j \in S}\{\tau(m_j)\}$$

Each queue Q_i stores not only messages that are already arrived (called *arrived* messages), but also an artificial *predicted* message at the end of Q_i. The timestamp of a *predicted* message is set based on the timestamp of the latest arrived message in Q_i and T_i^B. It is important to note that the selection procedure of the output message set is not solely based on the arrived messages but also considers the predicted messages, which can provide auxiliary information for the selection procedure. Nevertheless, a predicted message is never included in output message sets. Suppose there are currently k messages $\{m_i^1, ..., m_i^k\}$ in Q_i, m_i^k must be a predicted message and $m_i^1, ..., m_i^{k-1}$ are all arrived messages. The timestamp of m_i^k is set to be

$$\tau(m_i^k) = \tau(m_i^{k-1}) + T_i^B$$

When the system starts at time 0, a predicted message with timestamp 0 was initially put into each queue. Note that sometime a queue may only have a predicted message but no arrived message.

Definition 4 (Pivot)**.** *Let $S^1 = \{m_1^1, ..., m_N^1\}$, where each m_i^1 is the arrived message with the earliest timestamp in Q_i. The* pivot $m_{\mathbf{p}}$ *is the one with the* largest *timestamp among all elements in S^1. If several messages in S^1 all have the latest timestamp, the message with the maximum queue number is the pivot.*

The queue to which the pivot belongs is denoted as the pivot queue, while the remaining queues are denoted as the non-pivot queues. We use Λ to denote all the regular sets corresponding to the pivot $m_{\mathbf{P}}$. Note that the regular sets in Λ consist of messages currently in queues (either arrived or predicted) and must include $m_{\mathbf{P}}$. The *selected* set has the smallest time disparity among all regular sets in Λ.

Definition 5 (Selected Set)**.** *Let $m_{\mathbf{P}}$ be a pivot and Λ be the corresponding set of the regular sets that include $m_{\mathbf{P}}$. The* selected set *is the set that has the smallest time disparity among all elements in Λ. If multiple elements in Λ all have the smallest time disparity, the selected set $S = \{m_1, ..., m_N\}$ must satisfy the following condition: there does* not *exist another regular set $S' = \{m'_1, ..., m'_N\}$ in Λ s.t. (i) $\Delta(S') = \Delta(S)$ and (ii) $\exists m_i \in S : \tau(m'_i) < \tau(m_i)$.*

A selected set can include both arrived messages and predicted messages. We call a selected set containing only arrived messages a *published set* (denoted as $S^{\mathbf{PUB}}$). The messages in a published set are called *published messages*. If the selected set contains any predicted messages, the synchronizer must wait for them to arrive. Intuitively, the predicted message(s) can be used to combine a regular set with a smaller time disparity compared with the current selected set. However, in the case of $T_i^W > T_i^B$, a message may arrive with a larger timestamp than predicted. If the difference between the actual and predicted timestamp is significant, the message cannot be included in a selected set as expected. Therefore, the synchronizer can waste some time waiting for messages to arrive, further contributing to passing latency or reaction latency. The insight here is that if the predicted timestamp is too large (e.g., the difference between the predicted timestamp and the timestamp of the pivot exceeds the worst-case time disparity of the published set), the synchronizer does not need to wait for the predicted message, thereby to avoid wasting time. We will explain more about the above insights as well as the aspects relevant to the passing latency and reaction latency with an illustrative example in Section III-B.

A. Synchronization Policy

When a new message m_i arrives, the synchronizer will invoke Algorithm 1. First, the last message (must be a predicted message) is discarded from Q_i (Line 1). Then, m_i is put into the end of Q_i, which then is followed by a new predicted message with timestamp $\tau(m_i)+T_i^B$ (Line 2-3). After that, the pivot is set (Line 5) once there is at least one arrived message in each queue. And a selected set can only be obtained (Line 7) if all predicted messages have timestamps greater than $\tau(m_{\mathbf{P}})$. If the selected set only contains arrived messages, it will be published and all published messages should be discarded from the queues. Additionally, the messages earlier than the published messages will also be discarded from the queues, which are not included in any published sets (Line 8-11). Otherwise, if a selected set contains one or several predicted messages, Algorithm 1 exits immediately to wait for the predicted message(s) to arrive.

Algorithm 1: Synchronization Policy

```
   Input: the newly arrived message m_i
1  discard the last message in Q_i;
2  put m_i to the end of Q_i;
3  generate a predicted message with timestamp
     τ(m_i) + T_i^B and put it to the end of Q_i ;
4  while each queue has at least one arrived message do
5      m_P ← the current pivot (Definition 4);
6      if all predicted messages' timestamps > τ(m_P)
         then
7          S ← the selected set (Definition 5) ;
8          if all messages in S are arrived messages then
9              publish S;
10             for each m_j ∈ S do
11                 discard m_j and all messages before m_j
                     in the corresponding Q_j;
12         else
13             return;
14     else
15         return;
16 return;
```

We assume that the time required by Algorithm 1 to identify a selected set is negligible, i.e., it is considered to be 0. This assumption is made to simplify our analysis and to focus solely on the latency caused by waiting for messages to arrive and also the discarded messages. Furthermore, we use $\overline{\Delta}$ to represent the upper bound of time disparity for any published set, which is equal to the RHS of (12) in [3].

B. An Illustrative Example

We use Fig. 3 to illustrate Algorithm 1. The x-axis represents the timestamp and the messages' arrival time is not explicitly depicted in the figure. The downward arrows represent the messages buffered in the queues. $T_1^B = 3, T_2^B = 5, T_3^B = 10$ and $T_1^W = T_2^W = T_3^W = +\infty$.

At some time point, a message with timestamp 0 arrives in Q_3 and is set as the pivot as shown in Fig. 3-(a). The message set $\{m_1^1, m_2^1, m_3^1\}$ in Fig. 3-(a) is the first published set and the corresponding published messages will be discarded from the queues. Then, a message with timestamp 10 arrives at Q_3, which is set as the new pivot as shown in Fig. 3-(b). Please note that the indexes of messages are automatically updated in Algorithm 1 after discarding messages. For example, from Fig. 2-(a) to Fig. 2-(b), the notation of the message with timestamp of 3 in Q_1 is updated from m_1^2 to m_1^1 after m_1^1 with timestamp 0 is discarded.

The regular set $\{m_1^3, m_2^2, m_3^1\}$ in Fig. 3-(b) has the minimum time disparity, so it is the selected set. However, it cannot be published since m_2^2 is a predicted message. So the synchronizer will wait for m_2^2 to arrive. At some later point, m_1^4 and m_2^2 arrive successively as illustrated in Fig. 3-(c). Then the selected set $\{m_1^3, m_2^2, m_3^1\}$ in Fig. 3-(c) will be published

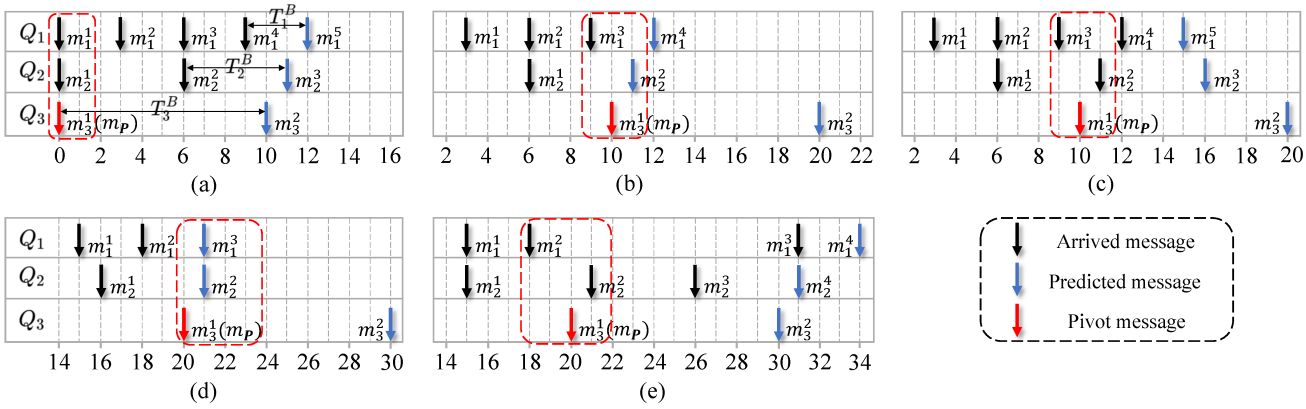


Fig. 3: An example illustrating the synchronization policy, where the x-axis represents the timestamp not the elapsed time. Each selected set is contained in the red dotted box.

since it only contains arrived messages. Furthermore, in Fig. 3-(c), it can be observed that m_1^3 arrived in Q_1 quite some time ago; however, it still had to wait for the arrival of both m_2^2 and m_3^1, resulting in a passing latency for m_1^3 until it could be published. Meanwhile, the messages m_1^1 and m_1^2 in Q_1 are not included in any published set, leading to an extra latency, which is part of the reaction latency for m_1^3 until it is published.

At some point, the queue status is shown in Fig. 3-(d), and m_3^1 is the pivot. The regular set $\{m_1^3, m_2^2, m_3^1\}$ in Fig. 3-(d) is the selected set with two predicted messages m_1^3 and m_2^2, so the synchronizer will wait for them to arrive. However, m_1^3 arrives with a timestamp 31, which is larger than predicted (i.e., 21 in this case) as shown in Fig. 3-(e). The regular set $\{m_1^2, m_2^2, m_3^1\}$ in Fig. 3-(e) is the selected set, so it will be published. In this case, m_1^3 arrives with a timestamp larger than predicted and is not included in the selected set, which further increases the passing latency and reaction latency (we will further discuss this in Section IV) for the published message m_1^2 in Fig. 3-(e).

IV. Passing Latency Upper Bound Analysis

In this section, we present the derivation of the upper bound of the passing latency for a published message $m_i^{\rho(i)}$ in Q_i ($i \in [1, N]$), which is included in a published set $S^{\mathbf{PUB}}$.

A. Passing Latency Analysis

According to Definition 1, we should upper bound the publishing time t_f for the published set $S^{\mathbf{PUB}}$.

Lemma 1. *Let $S^{\mathbf{PUB}}$ be the published set published at time t_f. t_f must be the arrival time of a message.*

Proof. Once a new message from any channel arrives, the synchronizer will invoke Algorithm 1. Recall that we assume the time for Algorithm 1 to find a published set is 0. Hence, the publishing time of a published set must be equal to the arrival time of the message that triggers the execution of Algorithm 1. The lemma is proved. □

Definition 6 (Latest Arrived Message)**.** *Let $S^{\mathbf{PUB}}$ be the published set. The latest arrived message $m_{\mathbf{L}} \in S^{\mathbf{PUB}}$ is the one with the latest arrival time among all messages in $S^{\mathbf{PUB}}$, i.e., $\forall m \in S^{\mathbf{PUB}}$: $\alpha(m_{\mathbf{L}}) \geq \alpha(m)$. Without loss of generality, let $m_{\mathbf{L}}$ come from Q_l ($l \in [1, N]$).*

Intuitively, the Algorithm 1 is expected to publish a published set upon the arrival of the latest arrived message. However, this is not always the case due to the utilization of predicted messages. Predicted messages can offer additional information during the selection process, thereby affecting the final publishing time of the published set. The insight is that the greater the difference between T_i^B and T_i^W, the longer the potential waiting time for the predicted message to arrive until the publishing time. Therefore, when a predicted message arrives with a later timestamp than predicted, it is possible that Algorithm 1 would not obtain a better selected set as expected. Consequently, the predicted message can further increase the passing latency.

Below, we first prove that the publishing time of a published set can be later than the arrival of the latest arrived message.

Lemma 2. *Let $S^{\mathbf{PUB}}$ be the published set returned by Algorithm 1 at time t_f, and $m_{\mathbf{L}} \in S^{\mathbf{PUB}}$ be the latest arrived message. Then, $t_f \geq \alpha(m_{\mathbf{L}})$ must hold.*

Proof. We prove this by contradiction, assuming $t_f < \alpha(m_{\mathbf{L}})$. According to Lemma 1, t_f must be the arrival time of a message $m \in S^{\mathbf{PUB}}$, i.e., $t_f = \alpha(m)$. Hence, $\alpha(m) < \alpha(m_{\mathbf{L}})$ and $m_{\mathbf{L}}$ has not yet been received at time t_f. Therefore, it is not possible to publish $S^{\mathbf{PUB}}$ at t_f. This contradicts the fact that $S^{\mathbf{PUB}}$ is published at t_f. So, $t_f \geq \alpha(m_{\mathbf{L}})$ must hold. □

Lemma 2 provides a lower bound for t_f. In the case of $t_f > \alpha(m_{\mathbf{L}})$, a message arrives at t_f and then the published set is published. However, this message is not included in this published set. To upper-bound the publishing time t_f, we first introduce the *earliest stable time*, which indicates the earliest time that each non-pivot queue contains at least one arrived message with a timestamp larger than the pivot.

Definition 7 (Earliest Stable Time)**.** *Let $S^{\mathbf{PUB}}$ be the published set and $m_{\mathbf{P}}$ be the corresponding pivot. The* earliest

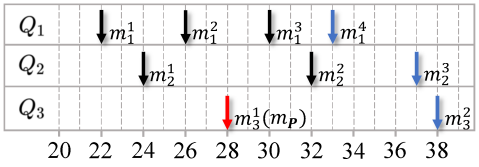


Fig. 4: Illustration of the earliest stable time.

stable time t^* *is the earliest time at which each non-pivot queue contains at least one arrived message with a timestamp larger than* $\tau(m_\mathbf{P})$.

If the pivot $m_\mathbf{P}$ *arrives after all those messages with timestamp greater than* $\tau(m_\mathbf{P})$*, the earliest stable time is* $t^* = \alpha(m_\mathbf{P})$.

According to the definition of t^*, it must be the arrival time of a message arriving at the queue. Let m_e^* denote the message arriving in Q_e ($e \in [1, N]$) at time t^*, i.e., $\alpha(m_e^*) = t^*$. As an example, in Fig. 4, suppose $\alpha(m_2^2) > \alpha(m_1^3) > \alpha(m_3^1)$, the earliest stable time is $t^* = \alpha(m_2^2)$. If $\alpha(m_3^1) > \alpha(m_2^2)$ and $\alpha(m_3^1) > \alpha(m_1^3)$, the earliest stable time would be $t^* = \alpha(m_3^1)$.

Following lemma proves that the worst-case publishing time t_f can not be later than the earliest stable time t^*.

Lemma 3. *For the published set* $S^{\mathbf{PUB}}$*, let* $m_\mathbf{P}$ *be the pivot,* t_f *be the publishing time, and* t^* *be the earliest stable time, then* $t_f \leq t^*$ *must hold.*

Proof. We prove this by contradiction, assuming $t_f > t^*$. By the definition of t^*, the while-condition in Line 4 and the if-condition in Line 6 in Algorithm 1 are both true. At time t^*, there must exist a selected set S returned from Line 7 of Algorithm 1. Since $S^{\mathbf{PUB}}$ is not published at t^*, S must not be $S^{\mathbf{PUB}}$. Since each non-pivot queue at least contains an arrived message with a timestamp later than $\tau(m_\mathbf{P})$, the predicted message in each queue must be "further away" from $m_\mathbf{P}$ than this arrived message. Therefore, S does not contain any predicted message and it must be a published set. As the same arrived message cannot be included in two published sets, $m_\mathbf{P}$ is not in S. Hence, for the pivot queue, S includes a message after $m_\mathbf{P}$ (note that $m_\mathbf{P}$ is the message with the earliest timestamp in the pivot queue). After S is published, all messages before the published message in S are discarded, and in particular, $m_\mathbf{P}$ is discarded, which contradicts that $m_\mathbf{P}$ is in $S^{\mathbf{PUB}}$ which is published after t^*. Therefore, the assumption is incorrect. $S^{\mathbf{PUB}}$ is published no later than t^*, i.e., $t_f \leq t^*$. □

Lemma 4. *Let* $S^{\mathbf{PUB}}$ *be any arbitrary published set, and* $m_\mathbf{P} \in S^{\mathbf{PUB}}$ *be the pivot. If* m_e^* *is the message arrives at* t^**, then* $\tau(m_e^*) \leq \tau(m_\mathbf{P}) + T_e^W$ *holds.*

Proof. By the definition of t^*, m_e^* can be the first arrived message in Q_e with timestamp larger than $\tau(m_\mathbf{P})$ or m_e^* is exactly $m_\mathbf{P}$. If m_e^* is $m_\mathbf{P}$, the lemma is obviously true. Otherwise, the last message before m_e^* in Q_e has the maximum timestamp value equal to $\tau(m_\mathbf{P})$ (since m_e^* is the first message with a timestamp later than $\tau(m_\mathbf{P})$). Hence, we must have $\tau(m_e^*) \leq \tau(m_\mathbf{P}) + T_e^W$. The lemma can be proved. □

Recall that we use $\overline{\Delta}$ to denote the upper bound of time disparity for any published set, which is equal to the RHS of (12) in [3], i.e.,

$$\overline{\Delta} = \max_{2 \leq n \leq N} \left\{ \frac{1}{n} \sum_{n-1 \ \ largest} T_j^W \right\}$$

Theorem 1 (**Passing Latency Upper Bound 1**). *Let* $S^{\mathbf{PUB}}$ *be any arbitrary published set. The passing latency experienced by* $m_i^{\rho(i)} \in S^{\mathbf{PUB}}$ *is upper-bounded by*

$$\overline{\Delta} + \mathcal{M} - D_i^B \quad \textit{where} \quad \mathcal{M} = \max_{j \in [1,N]} \left\{ T_j^W + D_j^W \right\} \tag{1}$$

Proof. By (1), Lemma 3, and Lemma 4, we have

$$\begin{aligned} & t_f - \alpha(m_i^{\rho(i)}) \leq \ t^* - \alpha(m_i^{\rho(i)}) = \alpha(m_e^*) - \alpha(m_i^{\rho(i)}) \\ \leq \ & (\tau(m_e^*) + D_e^W) - (\tau(m_i^{\rho(i)}) + D_i^B) \\ = \ & (\tau(m_\mathbf{P}) - \tau(m_i^{\rho(i)})) + (\tau(m_e^*) - \tau(m_\mathbf{P})) + D_e^W - D_i^B \\ \leq \ & \overline{\Delta} + (T_e^W + D_e^W) - D_i^B \leq \ \overline{\Delta} + \mathcal{M} - D_i^B \end{aligned}$$

The theorem is proved. □

The above upper bound in Theorem 1 could be pessimistic. The pessimism mainly comes from the term $\mathcal{M}$ in inequality (1). The insight here is that not all predicted messages with timestamps later than $\tau(m_\mathbf{P})$ will be waited to arrive. First, by Lemma 3, we know that the published set $S^{\mathbf{PUB}}$ must be published no later than t^*. In other words, before publishing $S^{\mathbf{PUB}}$, the synchronizer will only wait at least one predicted message with a timestamp later than $\tau(m_\mathbf{P})$ to arrive in each queue. The above analysis considers the worst-case publishing time, in which case at least one predicted message with a timestamp later than $\tau(m_\mathbf{P})$ should arrive in each queue. According to the generation of the predicted message, we know that the timestamp of a predicted message in Q_j is generated based on the minimum timestamp difference T_j^B. When T_j^B is too large, the difference between the timestamp of the pivot and the predicted message would exceed $\overline{\Delta}$. In this case, Algorithm 1 should not wait for this predicted message to arrive, since for the pivot $m_\mathbf{P}$ any selected set including this predicted message must have a time disparity larger than $\overline{\Delta}$ (note that the time disparity of the published set for any pivot $m_\mathbf{P}$ is at most $\overline{\Delta}$).

According to these observations, it is possible to add certain constraints on the term $\mathcal{M}$ in terms of T_j^B to mitigate the degree of pessimism, which allows us to enhance the accuracy of the computation for the upper bound on the passing latency. The following section will explain more clearly with an example and introduce a more optimistic upper bound for the passing latency.

B. The Second Upper Bound for Passing Latency

Below, we derive the second upper bound for the passing latency by dividing the published set $S^{\mathbf{PUB}}$ into three cases[3]:

- **Case 1**: all published messages in $S^{\mathbf{PUB}}$ from non-pivot queues have timestamp later than $\tau(m_{\mathbf{P}})$.
- **Case 2**: all published messages in $S^{\mathbf{PUB}}$ from non-pivot queues have timestamp earlier than $\tau(m_{\mathbf{P}})$.
- **Case 3**: some published messages in $S^{\mathbf{PUB}}$ have timestamp earlier than $\tau(m_{\mathbf{P}})$, while others' timestamp is later than $\tau(m_{\mathbf{P}})$.

We begin by demonstrating that in **Case 1**, the term $\mathcal{M}$ in inequality (1) can be simplified to only account for the delay element D_j^W, without the need to consider the minimal timestamp difference element T_j^W.

Lemma 5. *Let $S^{\mathbf{PUB}}$ be any arbitrary published set and $m_{\mathbf{P}} \in S^{\mathbf{PUB}}$ be the pivot. If $S^{\mathbf{PUB}}$ falls into **Case 1**, the passing latency experienced by $m_i^{\rho(i)} \in S^{\mathbf{PUB}}$ is upper-bounded by*

$$\mathcal{U}_p^1 = \overline{\Delta} + \mathcal{M}_1 - D_i^B, \quad \textit{where} \quad \mathcal{M}_1 = \max_{j\in[1,N]} \left\{D_j^W\right\} \tag{2}$$

Proof. The published message in each non-pivot queue must be the first message with a timestamp later than $\tau(m_{\mathbf{P}})$. When the latest arrived message $m_{\mathbf{L}} \in S^{\mathbf{PUB}}$ arrives, a selected set can be obtained from Line 7 in Algorithm 1. This selected set does not contain any predicted messages and must be the published set $S^{\mathbf{PUB}}$. So we have $t_f = \alpha(m_{\mathbf{L}})$. Since both $m_{\mathbf{L}}$ and $m_{\mathbf{P}}$ are included in $S^{\mathbf{PUB}}$, we can have $\tau(m_{\mathbf{L}}) \leq \tau(m_{\mathbf{P}}) + \overline{\Delta}$. In the worst-case, $m_i^{\rho(i)}$ can be $m_{\mathbf{P}}$. So, we have

$$\begin{aligned} t_f - \alpha(m_i^{\rho(i)}) =&\ \alpha(m_{\mathbf{L}}) - \alpha(m_i^{\rho(i)}) \\ \leq&\ (\tau(m_{\mathbf{L}}) + D_l^W) - (\tau(m_i^{\rho(i)}) + D_i^B) \\ \leq&\ \tau(m_{\mathbf{L}}) - \tau(m_{\mathbf{P}}) + D_l^W - D_i^B \\ \leq&\ \overline{\Delta} + D_l^W - D_i^B \leq\ \overline{\Delta} + \mathcal{M}_1 - D_i^B \end{aligned}$$

The lemma is proved. □

The key insight learned from Lemma 5 is that in **Case 1**, the latest arrived message is precisely the last one that Algorithm 1 should wait for, leading to the published set being obtained at the time $t_f = \alpha(m_{\mathbf{L}})$. In this case, the algorithm only needs to wait for all published messages included in the published set to arrive. However, in **Case 2** or **Case 3**, there is a possibility that Algorithm 1 may need to wait for additional messages to arrive, even if the latest arrived message $m_{\mathbf{L}}$ has already been arrived (i.e., all published messages have already arrived). More specifically, certain predicted messages have the potential to be included a selected set with a smaller time disparity. If these messages arrive with the predicted timestamps, the time disparity of the published set could be reduced. However, it is possible that they may arrive with timestamps that are later than predicted, thus disqualifying them from being included in a selected set, thereby prolonging the passing latency. In this case, the publishing time $t_f > \alpha(m_{\mathbf{L}})$ must hold. For example, in Fig. 3-(d), m_1^3 is a predicted message with a timestamp of 21. The selected set is $\{m_1^3, m_2^2, m_3^1\}$. Suppose that m_2^2 and m_2^3 arrive at Q_2, and then m_1^3 arrives with a timestamp of 31, as shown in Fig. 3-(e). Therefore, the published set will be $\{m_1^2, m_2^2, m_3^1\}$, and we have $t_f = \alpha(m_1^3) > \alpha(m_{\mathbf{L}}) = \alpha(m_2^2)$. Actually, if $T_1^B \geq 4$, Algorithm 1 would not wait for m_1^3 to arrive.

In both **Case 2** and **Case 3**, the challenge is to identify and exclude those predicted messages that will not be waited for (after $m_{\mathbf{L}}$ arrives), so that we can reduce the pessimism in the analysis of passing latency. Below, we first introduce how to do this by adding constraints on the minimal timestamp difference T_j^B in Lemmas 7 and 8 for **Case 2** and **Case 3**. Then we analyze how to incorporate these constraints into the term $\mathcal{M}$ for **Case 2** (in Lemma 9) and **Case 3** (in Lemma 10).

Lemma 6. *Let $m_{\mathbf{P}}$ be any pivot, and S be a selected set corresponding to $m_{\mathbf{P}}$. If $m_j \in S$ is a predicted message in Q_j ($j \in [1, N]$), then it satisfies:*

- $\tau(m_j) > \tau(m_{\mathbf{P}})$, ***and***
- $\nexists m_j'$: $\tau(m_{\mathbf{P}}) < \tau(m_j') < \tau(m_j)$.

Proof. By line 6 of Algorithm 1, S can only be obtained when all predicted messages have timestamps later than $\tau(m_{\mathbf{P}})$. m_j is a predicted message so $\tau(m_j) > \tau(m_{\mathbf{P}})$ must hold. We can assume that there exist m_j' in Q_j such that $\tau(m_{\mathbf{P}}) < \tau(m_j') < \tau(m_j)$. Therefore, m_j' must be an arrived message and $\tau(m_j') - \tau(m_{\mathbf{P}}) < \tau(m_j) - \tau(m_{\mathbf{P}})$. We can construct a regular set S' with all messages in S, replacing only m_j with m_j'. Then, we have $\Delta(S') \leq \Delta(S)$, which contradicts the fact that S is a selected set. The lemma is proved. □

In the following, we analyze the constraints on T_j^B under the context that for any pivot $m_{\mathbf{P}}$, the corresponding published set $S^{\mathbf{PUB}}$ falls into **Case 2** or **Case 3**, $m_{\mathbf{L}} \in S^{\mathbf{PUB}}$ is the latest arrived message, and S is a selected set obtained by Algorithm 1 *not earlier* than $\alpha(m_{\mathbf{L}})$.

Lemma 7. *If $m_j^{\rho(j)} \in S$ be a predicted message in Q_j ($j \in [1, N]$). Then, $T_j^B \leq 2\overline{\Delta}$.*

Proof. We prove this by contradiction, assuming $T_j^B > 2\overline{\Delta}$. By Lemma 6, $m_j^{\rho(j)-1}$ is an arrived message satisfying:

$$\tau(m_j^{\rho(j)-1}) \leq \tau(m_{\mathbf{P}}) \tag{3}$$

Since $S^{\mathbf{PUB}}$ falls into **Case 2** or **Case 3**, the published message (it must be an arrived message) $m_j^{\rho(j)-x} \in S^{\mathbf{PUB}}$ ($x \in \mathbb{N}^+$) must satisfy:

$$\tau(m_{\mathbf{P}}) - \overline{\Delta} \leq \tau(m_j^{\rho(j)-x}) \leq \tau(m_j^{\rho(j)-1}) \tag{4}$$

The predicted message $m_j^{\rho(j)}$ has a timestamp $\tau(m_j^{\rho(j)}) = \tau(m_j^{\rho(j)-1}) + T_j^B$. Combining it with (4) and (3), we have $\tau(m_j^{\rho(j)}) - \tau(m_{\mathbf{P}}) > \overline{\Delta}$. Therefore, any regular set containing $m_j^{\rho(j)}$ and $m_{\mathbf{P}}$ will have a time disparity greater than $\overline{\Delta}$. Since

[3]We emit the cases that the timestamp of a predicted message exactly equals $\tau(m_{\mathbf{P}})$ to simplify the presentation of the following proofs. This does not compromise the generality of our analysis, since we can add (or subtract) an infinitesimal value to (or from) its timestamp to fit our analysis.

S is obtained not earlier than $\alpha(m_{\mathbf{L}})$, there must exist a regular set that contains only arrived messages (which actually is the published set for $m_{\mathbf{P}}$) and it has a time disparity less than $\overline{\Delta}$. So, the synchronizer will not wait for $m_j^{\rho(j)}$ to arrive in any case, i.e., $m_j^{\rho(j)}$ can not be included in the selected set S, which contradicts the prerequisite $m_j^{\rho(j)} \in S$. Therefore, our assumption is incorrect and $T_j^B \leq 2\overline{\Delta}$ must hold. □

Lemma 7 states that in both **Case 2** and **Case 3**, the selected set obtained not earlier than $\alpha(m_{\mathbf{L}})$ for a given pivot can only include the predicted messages from the non-pivot queue Q_j that satisfies the condition $T_j^B \leq 2\overline{\Delta}$. These predicted messages will be waited to arrive until the publishing time. It is noted that the above condition is necessary but not sufficient.

Lemma 8. *If $m_j^{\rho(j)} \in S$ be a predicted message in Q_j ($j \in [1, N]$) and $\overline{\Delta} \leq T_j^B \leq 2\overline{\Delta}$, $\tau(m_j^{\rho(j)-1})$ must satisfy*[4]*:*

$$\tau(m_j^{\rho(j)-1}) \leq \tau(m_{\mathbf{P}}) + \overline{\Delta} - T_j^B \tag{5}$$

Proof. Since the predicted message $m_j^{\rho(j)}$ is included in S, $\tau(m_j^{\rho(j)}) - \tau(m_{\mathbf{P}})$ must not be large than $\overline{\Delta}$, i.e., $\tau(m_j^{\rho(j)}) \leq \tau(m_{\mathbf{P}}) + \overline{\Delta}$. Since $\tau(m_j^{\rho(j)}) \geq \tau(m_j^{\rho(j)-1}) + T_j^B$, we have $\tau(m_j^{\rho(j)-1}) \leq \tau(m_{\mathbf{P}}) + \overline{\Delta} - T_j^B$. Proved. □

Lemma 8 reveals that for any pivot, when $\overline{\Delta} \leq T_j^B \leq 2\overline{\Delta}$, the predicted messages in Q_j can be included into a selected set and then be waited for arrival before the selected set for this pivot can be published, but only if the condition specified in Eq. (5) is satisfied.

To derive the upper bound for **Case 2** and **Case 3**, we first introduce some auxiliary notations. For any pivot $m_{\mathbf{P}}$, the time disparity of its published set is upper-bounded by $\overline{\Delta}$. We define two sets for the queue index j:

$$\phi_1 = \left\{ j | j \in [1, N] \wedge 0 < T_j^B < \overline{\Delta} \right\}$$

$$\phi_2 = \left\{ j | j \in [1, N] \wedge \overline{\Delta} \leq T_j^B \leq 2\overline{\Delta} \right\}$$

Lemma 9. *Let $S^{\mathbf{PUB}}$ be any arbitrary published set and $m_{\mathbf{P}} \in S^{\mathbf{PUB}}$ be the pivot. If $S^{\mathbf{PUB}}$ falls into **Case 2**, the passing latency experienced by $m_i^{\rho(i)} \in S^{\mathbf{PUB}}$ is upper-bounded by*

$$\mathcal{U}_p^2 = \overline{\Delta} + \mathcal{M}_2 - D_i^B \tag{6}$$

where

$$\mathcal{M}_2 = \max\left\{ \mathcal{M}_2^1, \mathcal{M}_2^2 \right\}$$

$$\mathcal{M}_2^1 = \max_{j \in \phi_1} \left\{ T_j^W + D_j^W \right\}$$

$$\mathcal{M}_2^2 = \max_{j \in \phi_2} \left\{ \overline{\Delta} - T_j^B + T_j^W + D_j^W \right\}$$

Proof. Suppose that before publishing $S^{\mathbf{PUB}}$, $m_j^{\rho(j)}$ ($j \in [1, N]$) be the first message, that arrives in Q_j, with a timestamp of $\tau(m_j^{\rho(j)}) > \tau(m_{\mathbf{P}})$. By Lemma 7, the synchronizer waits for $m_j^{\rho(j)}$ to arrive only if $T_j^B \leq 2\overline{\Delta}$. When $j \in \phi_2$, by Lemma 8, the message $m_j^{\rho(j)-1}$ has a maximum timestamp value of $\tau(m_j^{\rho(j)-1}) = \tau(m_{\mathbf{P}}) + \overline{\Delta} - T_j^B$. Therefore, the timestamp of $m_j^{\rho(j)}$ (when it arrives) must satisfy

$$\tau(m_j^{\rho(j)}) \leq \tau(m_{\mathbf{P}}) + \overline{\Delta} - T_j^B + T_j^W \tag{7}$$

When $j \in \phi_1$, the message $m_j^{\rho(j)-1}$ has a timestamp not later than $\tau(m_{\mathbf{P}})$. In the worst case, we have

$$\tau(m_j^{\rho(j)}) \leq \tau(m_{\mathbf{P}}) + T_j^W \tag{8}$$

Suppose $m_h^{\rho(h)}$ ($h \in \phi_1 \cup \phi_2$) be the first message with a timestamp larger than $\tau(m_{\mathbf{P}})$ in Q_h, and let it be the last one to arrive among all such messages in the queues. Based on Eqs. (7) and (8), we can derive

$$\begin{aligned}
& t_f - \alpha(m_i^{\rho(i)}) \leq \ \alpha(m_h^{\rho(h)}) - \alpha(m_i^{\rho(i)}) \\
\leq\ & (\tau(m_h^{\rho(h)}) + D_h^W) - (\tau(m_i^{\rho(i)}) + D_i^B) \\
=\ & (\tau(m_{\mathbf{P}}) - \tau(m_i^{\rho(i)})) + (\tau(m_h^{\rho(h)}) - \tau(m_{\mathbf{P}})) + D_h^W - D_i^B \\
\leq\ & \overline{\Delta} + \max\left\{ \mathcal{M}_2^1, \mathcal{M}_2^2 \right\} - D_i^B
\end{aligned}$$

Proved. □

Lemma 10. *Let $S^{\mathbf{PUB}}$ be any arbitrary published set and $m_{\mathbf{P}} \in S^{\mathbf{PUB}}$ be the pivot. If $S^{\mathbf{PUB}}$ falls into **Case 3**, the passing latency experienced by $m_i^{\rho(i)} \in S^{\mathbf{PUB}}$ is upper-bounded by*

$$\mathcal{U}_p^2 = \overline{\Delta} + \mathcal{M}_2 - D_i^B$$

Proof. Let $m_j^{\rho(j)}$ ($j \in [1, N]$) be the first message that arrives in Q_j before publishing $S^{\mathbf{PUB}}$ and $\tau(m_j^{\rho(j)}) > \tau(m_{\mathbf{P}})$. Let $\tau(m_{\mathbf{P}}) + \sigma$ ($0 < \sigma < \overline{\Delta}$) be the timestamp of the published message that has the latest timestamp among all messages in $S^{\mathbf{PUB}}$. By Lemma 7, $T_j^B \leq 2\overline{\Delta}$ must hold. Similarly, we have $\tau(m_j^{\rho(j)}) \leq \tau(m_{\mathbf{P}}) + \overline{\Delta} - T_j^B + T_j^W$ if $j \in \phi_2$. And we have $\tau(m_j^{\rho(j)}) \leq \tau(m_{\mathbf{P}}) + T_j^W$ if $j \in \phi_1$. Suppose $m_h^{\rho(h)}$ ($h \in \phi_1 \cup \phi_2$) is the first message with a timestamp larger than $\tau(m_{\mathbf{P}})$ in Q_h, and let it be the last one to arrive among all such messages in the queues. We have

$$\begin{aligned}
& t_f - \alpha(m_i^{\rho(i)}) \leq \ \alpha(m_h^{\rho(h)}) - \alpha(m_i^{\rho(i)}) \\
\leq\ & (\tau(m_h^{\rho(h)}) + D_h^W) - (\tau(m_i^{\rho(i)}) + D_i^B) \\
=\ & (\tau(m_{\mathbf{P}}) + \sigma - \tau(m_i^{\rho(i)})) + (\tau(m_h^{\rho(h)}) - \tau(m_{\mathbf{P}}) - \sigma \\
& + D_h^W) - D_i^B \\
\leq\ & \overline{\Delta} + \max\left\{ \mathcal{M}_2^1 - \sigma, \mathcal{M}_2^2 - \sigma \right\} - D_i^B \\
<\ & \overline{\Delta} + \mathcal{M}_2 - D_i^B
\end{aligned}$$

In conclusion, the lemma is proved. □

Theorem 2 (Passing Latency Upper Bound 2). *Let $S^{\mathbf{PUB}}$ be any arbitrary published set. The passing latency experienced by $m_i^{\rho(i)} \in S^{\mathbf{PUB}}$ is upper-bounded by*

$$\mathcal{U}_p = \max\left\{ \mathcal{U}_p^1, \mathcal{U}_p^2 \right\} \tag{9}$$

[4]Of course, there exists a minimal limit as well, i.e., $\tau(m_j^{\rho(j)-1}) > \tau(m_{\mathbf{P}}) - \overline{\Delta}$. However, our focus here lies on the maximum limit.

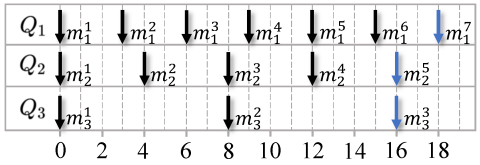


Fig. 5: Illustration of the continuous published sets.

where $\mathcal{U}_p^1$ and $\mathcal{U}_p^2$ are defined in Eqs. (2) and (6), respectively.

Proof. The published set $S^{\mathbf{PUB}}$ must fall into one of the three cases, i.e., **Case 1**, **Case 2**, or **Case 3**. The upper bound of the passing latency can be obtained by choosing the maximum upper bound among the three cases. Thus, we reach the conclusion. □

V. Reaction Latency Upper Bound Analysis

This section analyzes the upper bound of the reaction latency for a published message $m_i^{\rho(i)}$ included in $S^{\mathbf{PUB}}$.

We say two messages $m_i^{\rho(i)}$ and $m_i^{\rho(i)+1}$ that both are from the same queue Q_i ($i \in [1, N]$) are continuous. Accordingly, we present the definition of continuous published sets:

Definition 8 (Continuous Published Sets)**.** *Let $S_1^{\mathbf{PUB}}$ and $S_2^{\mathbf{PUB}}$ be any two different published sets. Without loss of generality, suppose $S_1^{\mathbf{PUB}}$ is outputted before $S_2^{\mathbf{PUB}}$. $S_1^{\mathbf{PUB}}$ and $S_2^{\mathbf{PUB}}$ are continuous iff: $\exists m_i^{\rho(i)} \in S_1^{\mathbf{PUB}} \wedge m_i^{\rho(i)+1} \in S_2^{\mathbf{PUB}}$ ($i \in [1, N]$), $m_i^{\rho(i)}$ and $m_i^{\rho(i)+1}$ are continuous.*

For example, as shown in Fig. 5, the first published set is $S_1^{\mathbf{PUB}} = \{m_1^1, m_2^1, m_3^1\}$ and the second published set is $S_2^{\mathbf{PUB}} = \{m_1^4, m_2^3, m_3^2\}$. $S_1^{\mathbf{PUB}}$ and $S_2^{\mathbf{PUB}}$ are continuous since $m_3^1 \in S_1^{\mathbf{PUB}}$ and $m_3^2 \in S_2^{\mathbf{PUB}}$ are continuous.

Below, we first drive an upper bound for the time interval between the arrivals of two published messages that are in the same queue and included in two continuous published sets (Lemma 12). Then, we derive the upper bound for the reaction latency in Theorem 3.

Lemma 11. *Any two published sets $S_1^{\mathbf{PUB}}$ and $S_2^{\mathbf{PUB}}$, which are sequentially published by Algorithm 1, are continuous.*

Proof. Suppose that $S_2^{\mathbf{PUB}}$ is published after $S_1^{\mathbf{PUB}}$. By Algorithm 1 Line 10-11, when publishing $S_1^{\mathbf{PUB}}$, each message $m_j \in S_1^{\mathbf{PUB}}$ and all messages before m_j will be discarded from Q_j. After publishing $S_1^{\mathbf{PUB}}$, there must be no discarded message after m_j before publishing $S_2^{\mathbf{PUB}}$ (since no message overflow occurs and messages can only be discarded by Line 10-11 of Algorithm 1). Then, a new pivot (it must be included in $S_2^{\mathbf{PUB}}$) will be selected for $S_2^{\mathbf{PUB}}$ among the arrived messages, each of which must have the earliest timestamp in the corresponding queue. And, each of these messages must be continuous to the published message in the corresponding queue, which is included in $S_1^{\mathbf{PUB}}$. Therefore, the new pivot of $S_2^{\mathbf{PUB}}$, which is one of these messages, must be continuous to a published message included in $S_1^{\mathbf{PUB}}$. By Definition 8, $S_1^{\mathbf{PUB}}$ and $S_2^{\mathbf{PUB}}$ must be continuous. □

Lemma 12. *Let $S_1^{\mathbf{PUB}}$ and $S_2^{\mathbf{PUB}}$ be any two published sets ($S_2^{\mathbf{PUB}}$ is published immediately after $S_1^{\mathbf{PUB}}$), and $m_i^{\rho(i)-x} \in S_1^{\mathbf{PUB}}$ ($x \in \mathbb{N}^+$) and $m_i^{\rho(i)} \in S_2^{\mathbf{PUB}}$. The latency $\alpha(m_i^{\rho(i)}) - \alpha(m_i^{\rho(i)-x})$ is upper-bounded by:*

$$\mathcal{U}_d = 2\overline{\Delta} + T_{max}^W + D_i^W - D_i^B \tag{10}$$

where

$$T_{max}^W = \max_{j \in [1,N]} \{T_j^W\}$$

Proof. According to Lemma 11, $S_1^{\mathbf{PUB}}$ and $S_2^{\mathbf{PUB}}$ must be continuous. Hence, there must exist two continuous messages $m_h^{\rho(h)-1} \in S_1^{\mathbf{PUB}}$ and $m_h^{\rho(h)} \in S_2^{\mathbf{PUB}}$ ($h \in [1, N]$). Hence, we have

$$\tau(m_h^{\rho(h)}) - \tau(m_h^{\rho(h)-1}) \leq T_h^W \leq T_{max}^W$$

In the worst case, $m_i^{\rho(i)-x}$ is the one with the earliest timestamp among all messages in $S_1^{\mathbf{PUB}}$, and $m_i^{\rho(i)}$ is the one with the latest timestamp among all messages in $S_2^{\mathbf{PUB}}$. Therefore, we can know $\tau(m_i^{\rho(i)}) - \tau(m_h^{\rho(h)}) \leq \overline{\Delta}$ and $\tau(m_h^{\rho(h)-1}) - \tau(m_i^{\rho(i)-x}) \leq \overline{\Delta}$. Combining the above inequalities, we have

$$\tau(m_i^{\rho(i)}) - \tau(m_i^{\rho(i)-x}) \leq 2\overline{\Delta} + T_{max}^W$$

Then, we can derive

$$\begin{aligned} & \alpha(m_i^{\rho(i)}) - \alpha(m_i^{\rho(i)-x}) \\ \leq\ & (\tau(m_i^{\rho(i)}) + D_i^W) - (\tau(m_i^{\rho(i)-x}) + D_i^B) \\ \leq\ & 2\overline{\Delta} + T_{max}^W + D_i^W - D_i^B \end{aligned}$$

The lemma is proved. □

Theorem 3 (**Reaction Latency Upper Bound**)**.** *Suppose that $S^{\mathbf{PUB}}$ is published at time t_f, $m_i^{\rho(i)} \in S^{\mathbf{PUB}}$, and $m_i^{\rho(i)-x}$ ($x \in \mathbb{N}^+$) is the last message before $m_i^{\rho(i)}$ that was selected into a published set. The reaction latency of a message $m_i^{\rho(i)}$ is upper-bounded by $\mathcal{U}_p + \mathcal{U}_d$, where $\mathcal{U}_p$ and $\mathcal{U}_d$ are defined in Eqs. (9) and (10), respectively.*

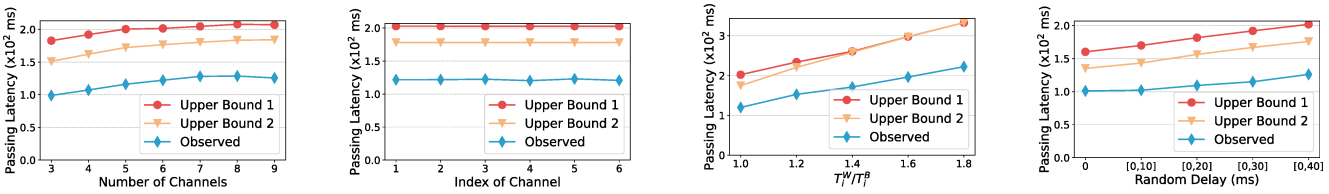
Proof. By Definition 2, the reaction latency of $m_i^{\rho(i)}$ is

$$t_f - \alpha(m_i^{\rho(i)-x}) = \{t_f - \alpha(m_i^{\rho(i)})\} + \{\alpha(m_i^{\rho(i)}) - \alpha(m_i^{\rho(i)-x})\}$$

where $t_f - \alpha(m_i^{\rho(i)})$ and $\alpha(m_i^{\rho(i)}) - \alpha(m_i^{\rho(i)-x})$ are upper-bounded by $\mathcal{U}_p$ and $\mathcal{U}_d$, respectively. The theorem is proved. □

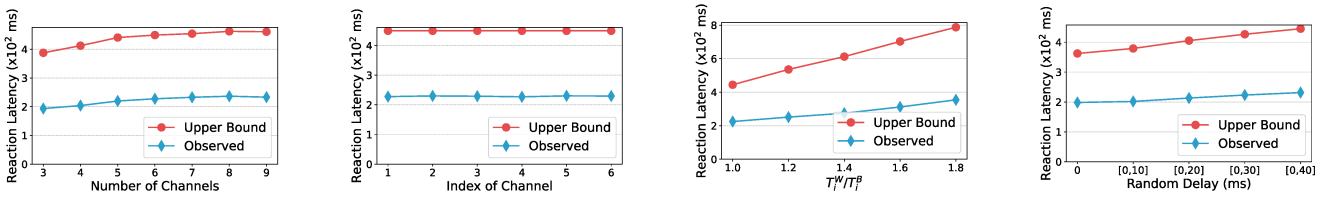
VI. Experiments

We conducted a series of experiments to evaluate the precision of our latency analysis (Theorem 1, 2 and 3), including both passing latency and reaction latency. All experiments were conducted on a desktop computer with an Intel(R) Core(TM) i7-10700 CPU running at 2.90GHz. The computer was installed with ROS 2, specifically the *Humble Hawksbill* version running on Ubuntu 20.04.4 LTS. The source code is available at *https://github.com/ruoxianglee/latency_analysis*.



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

Fig. 6: Passing latency evaluation results.



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

Fig. 7: Reaction latency evaluation results.

A. Experiment Setting

We perform experiments in real execution in ROS. To ensure precise evaluation of the timing behavior of messages, we propose a new timing data structure (as shown in Table I) for message tracing. This structure can be directly integrated into the pre-existing message types in ROS. It contains the message timestamp, arrival time, and publishing time. The assignment of the message timestamp occurs during the message generation process in the sensor data sampling task. The synchronizer assigns the arrival time to each incoming message upon its arrival and a publishing time to each message in the published set once the set is selected and ready for output. Based on the above timing information, we can trace messages received by the synchronizer, thereby enabling us to observe and analyze the passing latency and the reaction latency.

We run experiments to evaluate the latency experienced by messages in the synchronizer using artificial input messages generated by timers. The case-study system involves up to 9 timer callbacks, which can be configured, and a message synchronizer. Each timer can be configured with minimal and maximum timestamp difference (T_i^B and T_i^W) for artificial input message generation. Our objective is to observe the passing latency and the reaction latency for the messages that pass through the synchronizer. Therefore, to ensure accurate results, we have designed the case-study system as simple as possible, excluding other callbacks. Additionally, we employ a dedicated single-threaded executor for each timer callback and the synchronizer. With the above design, we can minimize any potential interference among callbacks caused by the ROS executor scheduling. In this system, we can configure the delay time experienced by each message before it arrives at the synchronizer, which helps us evaluate the performance of the synchronizer under different settings.

TABLE I: Timing Data Structure

Name	Type	Description
timestamp	`builtin_interfaces/Time`	Data sampling time
arrival_time	`builtin_interfaces/Time`	Message arrival time
publishing_time	`builtin_interfaces/Time`	Message publishing time

B. Latency Evaluation

We evaluate both passing latency and reaction latency with different settings, including the different number of input channels (from 3 to 9, as currently the ROS Message Filter supports up to 9 input channels), the ratio between T_i^W and T_i^B (chosen between 1 and 1.8), and different delay time before messages arrive at the synchronizer (chosen between 0 and 40 ms). In each experiment (which is corresponding to each x-value in Fig. 6 and 7), we record the worst-case latency among 5000 observations. Each point (in Fig. 6 and 7) corresponds to 100 experiments. The value of each point is determined by computing an average of the recorded worst-case latencies of all these 100 experiments. We conducted multiple experiments for each point because only one experiment with a specific setting is not representative. Therefore, different experiments with different settings are performed to guarantee the evaluation's generality.

We compare the values of **Upper Bound 1**, **Upper Bound 2** and **Observed** for the passing latency (Fig. 6-(a) to (d)), and the values of **Upper Bound** and **Observed** for the reaction latency (Fig. 7-(a) to (d)). These parameters are defined:

- **Upper Bound 1**: the first passing latency upper bound calculated by Eq. (1) in Theorem 1.
- **Upper Bound 2**: the second passing latency upper bound calculated by Eq. (9) in Theorem 2.
- **Upper Bound**: the reaction latency upper bound in Theorem 3.
- **Observed**: the maximum observed passing latency or reaction latency in real execution.

For the passing latency evaluation, Fig. 6-(a) shows the experiment results under the different number of channels (x-axis), where messages of each channel were generated periodically (i.e., $T_i^B = T_i^W$) with period randomly distributed in $[50, 100]$ and delay randomly distributed in $[1, 40]$. While it is possible to show the values of **Upper Bound 1**, **Upper Bound 2**, and **Observed** for all channels in each setting, we only illustrate the results for the first channel in Fig. 6-(a) (as well as in Fig. 6-(c) and (d)). However, we can certainly

reach the same evaluation conclusion for all other channels. To further demonstrate this, we also illustrate the results of all six channels in Fig. 6-(b) as an example, where we use the same setting as Fig. 6-(a), except that the number of channels was kept constant at 6. In Fig. 6-(c), the messages are no longer generated periodically, but with timestamp separation randomly distributed between T_i^B and T_i^W, and the ratio between T_i^W and T_i^B varies as indicated by the x-axis. In Fig. 6-(d), we use the same setting as in Fig. 6-(a), but set the number of channels to 6 and change the range of delay experienced by each message as shown by the x-axis.

For the reaction latency evaluation, we use the same setting in Fig. 7-(a), (b), (c) and (d) as Fig. 6-(a), (b), (c) and (d), respectively. Again, we only illustrate the results (values of **Upper Bound** and **Observed**) for the first channel in Fig. 7-(a) (as well as in Fig. 7-(c) and (d)). And an example with the results of all six channels is illustrated in Fig. 7-(b).

From the experiment results in Fig. 6-(a) to (d), we can see that our upper bounds for the passing latency (Theorem 1 and 2) have good precision. As depicted in Fig. 6-(c), as the ratio between T_i^W and T_i^B increases, particularly at ratios of 1.6 and 1.8), the difference between **Upper Bound 1** and **Upper Bound 2** becomes negligible. The reason is that as the ratio increases, it exacerbates the difference between T_i^B and T_i^W. Since $\overline{\Delta}$ is calculated based on T_i^W, $T_i^B > \overline{\Delta}$ can always hold, and the constraints introduced in the second upper bound for the passing latency become invalid. From the experiment results (Fig. 7-(a) to (d)), we can know that our upper bound for the reaction latency (Theorem 3) has a certain level of pessimism. Further analysis and refinement may be necessary to assess the upper bound accurately.

Based on the experiment results, we can observe that the synchronization policy can produce considerable latency. As illustrated in Fig. 6-(a), (c), and (d), the passing latency increases as the number of channels, the ratio between T_i^W and T_i^B, or the range of delay experienced by each message before arriving at the synchronizer becomes larger, in terms of **Upper Bound 1**, **Upper Bound 2** and **Observed**. As shown in Fig. 6-(b), the passing latency is almost the same across different channels, in terms of **Upper Bound 1**, **Upper Bound 2** and **Observed**. For the reaction latency, we can reach the same conclusion based on Fig. 7-(a) to (d).

VII. Related Work

Data fusion algorithms are commonly developed with the assumption that data from multiple sensors are perfectly aligned, although this is rarely the case in reality. To address this issue, various techniques have been proposed to compensate for the temporal inconsistency of input data [30]–[33], which only work when the temporal inconsistency falls within a certain range. Message synchronization before data fusion is a crucial component that warrants careful consideration and attention. Previous studies [2], [34]–[36] have focused on precisely timestamping sensor data in the context of multi-sensor data fusion. In this paper, we assume that sensor data has already been associated with valid timestamps in the same coordinate system using these existing techniques. Our focus is on the problem that arises after timestamping, i.e., the latency caused when managing the sensor data flows in the computing system based on these timestamps.

In recent years, some work has been conducted on formal real-time performance analysis of ROS2, such as exploring response time analysis by modeling execution of ROS2 applications as processing chains or a DAG [16], [17], [20], [22], [24] executing on the ROS2 default scheduler, i.e., the executor. [18], [21], [23], [24] proposed to address the limitations of the default scheduling strategy of ROS2 by enhancing or redesigning the executor. In [19], the authors propose an automatic latency manager that applies existing real-time scheduling theory to latency control of critical call-back chains in ROS2 applications. [14] proposed an end-to-end timing analysis for cause-effect chains in ROS2, considering the maximum end-to-end reaction time and maximum data age metrics. However, all of the research mentioned above focuses solely on the executor component in ROS2 only for the end-to-end latency (response time) analysis without considering the Message Synchronizer. [37] proposed a synchronization system implemented in a node to harmonize communication between nodes, which works similarly to the message synchronization policy in ROS. Recent work [3], [7] modeled the message synchronization policy in ROS and formally analyzed the worst-case time disparity of the output message set as well as the important properties of the policy. However, their analysis only focuses on the time disparity metric and neglects to consider the latency caused by the policy, which is closely tied to end-to-end latency and is a critical factor in the reaction time of the system as a whole.

Previous research on real-time scheduling and analysis has investigated various real-time performance metrics, including response time [38], [39], tardiness [40] and data freshness [10]–[13], [41]. However, these analysis techniques cannot be directly applied to ROS systems. Furthermore, analyzing latency associated with the message synchronization policy in ROS remains an open research question.

VIII. Conclusion

In this paper, we explore two types of latency metrics associated with the ROS message synchronization policy, i.e., the *passing latency* and the *reaction latency*, and formally analyze the upper bounds for both latency. We conduct experiments under different settings, including the different number of channels, the varied data sampling periods, and the random delay time experienced by messages before arriving at the synchronization policy, to evaluate the precision of our proposed latency upper bounds against the maximal observed latency in real execution. In the future, we plan to improve the design and implementation of the ROS message synchronization policy, considering both the time disparity and latency aspects, with the ultimate goal of achieving better real-time performance in ROS systems.

REFERENCES

[1] B. Yu, W. Hu, L. Xu, J. Tang, S. Liu, and Y. Zhu, "Building the computing system for autonomous micromobility vehicles: Design constraints and architectural optimizations," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 1067–1081.

[2] S. Liu, B. Yu, Y. Liu, K. Zhang, Y. Qiao, T. Y. Li, and et al., "The matter of time — a general and efficient system for precise sensor synchronization in robotic computing," in *RTAS*, 2021.

[3] R. Li, N. Guan, X. Jiang, Z. Guo, Z. Dong, and M. Lv, "Worst-Case Time Disparity Analysis of Message Synchronization in ROS," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 40–52.

[4] "ROS Introduction." [Online]. Available: http://wiki.ros.org/ROS/Introduction

[5] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[6] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[7] R. Li, Z. Dong, J.-M. Wu, C. J. Xue, and N. Guan, "Modeling and Property Analysis of the Message Synchronization Policy in ROS," in *2023 IEEE International Conference on Mobility: Operations, Services, and Technologies (MOST)*. IEEE, 2023.

[8] "Apollo Cyber RT." [Online]. Available: https://cyber-rt.readthedocs.io/en/latest/

[9] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 287–296.

[10] A. Davare, Q. Zhu, M. Di Natale, C. Pinello, S. Kanajan, and A. Sangiovanni-Vincentelli, "Period optimization for hard real-time distributed automotive systems," in *Proceedings of the 44th annual Design Automation Conference*, 2007, pp. 278–283.

[11] T. Kloda, A. Bertout, and Y. Sorel, "Latency analysis for data chains of real-time periodic tasks," in *2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA)*, vol. 1. IEEE, 2018, pp. 360–367.

[12] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-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, pp. 1–24, 2019.

[13] M. Günzel, K.-H. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J.-J. Chen, "Timing analysis of asynchronized distributed cause-effect chains," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 40–52.

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

[15] E. Peguero, M. Labrador, and B. Cook, "Assessing jitter in sensor time series from android mobile devices," in *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 2016, pp. 1–8.

[16] D. Casini, T. Blaß, I. Lütkebohle, and B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based scheduling," in *31st Euromicro Conference on Real-Time Systems*. Schloss Dagstuhl, 2019, pp. 1–23.

[17] Y. Tang, Z. Feng, N. Guan, X. Jiang, M. Lv, Q. Deng, and W. Yi, "Response time analysis and priority assignment of processing chains on ROS2 executors," in *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2020, pp. 231–243.

[18] H. Choi, Y. Xiang, and H. Kim, "PiCAS: New design of priority-driven chain-aware scheduling for ROS2," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 251–263.

[19] T. Blass, A. Hamann, R. Lange, D. Ziegenbein, and B. B. Brandenburg, "Automatic Latency Management for ROS 2: Benefits, Challenges, and Open Problems," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2021, pp. 264–277.

[20] T. Blaß, D. Casini, S. Bozhko, and B. B. Brandenburg, "A ROS 2 Response-Time Analysis Exploiting Starvation Freedom and Execution-Time Variance," in *2021 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2021, pp. 41–53.

[21] C. Randolph, "Improving the Predictability of Event Chains in ROS 2," Ph.D. dissertation, Master's thesis, Delft University of Technology. http://resolver. tudelft ..., 2021.

[22] X. Jiang, D. Ji, N. Guan, R. Li, Y. Tang, and Y. Wang, "Real-time scheduling and analysis of processing chains on multi-threaded executor in ros 2," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 27–39.

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

[24] H. Sobhani, H. Choi, and H. Kim, "Timing analysis and priority-driven enhancements of ros 2 multi-threaded executors."

[25] R. I. Davis, A. Burns, R. J. Bril, and J. J. Lukkien, "Controller area network (can) schedulability analysis: Refuted, revisited and revised," *Real-Time Systems*, vol. 35, pp. 239–272, 2007.

[26] J. Rox and R. Ernst, "Formal timing analysis of full duplex switched based ethernet network architectures," SAE Technical Paper, Tech. Rep., 2010.

[27] J. Diemer, J. Rox, and R. Ernst, "Modeling of ethernet avb networks for worst-case timing analysis," *IFAC Proceedings Volumes*, vol. 45, no. 2, pp. 848–853, 2012.

[28] "ROS Message Filter: Exact Time Synchronization Policy." [Online]. Available: http://wiki.ros.org/message_filters/ExactTime

[29] "ROS Message Filter: Approximate Time Synchronization Policy." [Online]. Available: http://wiki.ros.org/message_filters/ApproximateTime

[30] H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization," in *BMVC*, 2017.

[31] M. Huber, M. Schlegel, and G. Klinker, "Application of time-delay estimation to mixed reality multisensor tracking," in *J. Virtual Real. Broadcast.*, 2014.

[32] J. Peršić, L. Petrović, I. Marković, and I. Petrović, "Online multi-sensor calibration based on moving object tracking," in *Adv. Robotics*, 2021.

[33] M. R. Nowicki, "Spatio-temporal calibration of camera and 3d laser scanner," in *IEEE Robotics and Automation Letters*, 2020.

[34] D. J. Yeong, G. Velasco-Hernandez, J. Barry, J. Walsh *et al.*, "Sensor and sensor fusion technology in autonomous vehicles: A review," in *Sensor*, 2021.

[35] Z. Wang, Y. Wu, and Q. Niu, "Multi-sensor fusion in automated driving: A survey," in *IEEE Access*, 2016.

[36] J. Kelly and G. S. Sukhatme, "A general framework for temporal calibration of multiple proprioceptive and exteroceptive sensors," in *ISER*, 2010.

[37] Y. Saito, T. Azumi, S. Kato, and N. Nishio, "Priority and synchronization support for ROS," in *2016 IEEE 4th International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*. IEEE, 2016, pp. 77–82.

[38] M. Joseph and P. Pandya, "Finding response times in a real-time system," in *The Computer Journal*, 1986.

[39] L. Sha, T. Abdelzaher, A. Cervin, T. Baker, A. Burns, G. Buttazzo, and et al., "Real time scheduling theory: A historical perspective," in *Real Time Syst.*, 2004.

[40] U. Devi and J. Anderson, "Tardiness bounds under global edf scheduling on a multiprocessor," in *RTSS*, 2005.

[41] 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 *IEEE Real-Time Systems Symposium: 30/11/2009-03/12/2009*. IEEE Communications Society, 2009.