Modeling and Analysis of the LatestTime Message Synchronization Policy in ROS

Chenhao Wu, Ruoxiang Li, Naijun Zhan, and Nan Guan, *Member, IEEE*

***Abstract*—Sensor fusion plays a critical role in modern robotics and autonomous systems. In reality, the sensor data destined for the fusion algorithm may have substantially different sampling times. Without proper management, this could lead to poor sensor fusion quality. Robot operating system (ROS) is the most popular robotic software framework, providing essential mechanisms for synchronizing messages to mitigate timing inconsistencies during sensor fusion. Recently, ROS introduced a new *LatestTime* message synchronization policy. In this article, we formally model the behavior of the *LatestTime* policy and analyze its worst-case real-time performance. Our investigation uncovers a defect of the *LatestTime* policy that may cause infinite latency in publishing subsequent outputs. We propose a solution to address this defect and develop safe and tight upper bounds on worst-case real-time performance, in terms of both the maximal temporal inconsistency of its outputs and the incurred latency. Experiments are conducted to evaluate the precision, safety and robustness of our theoretical results.**

***Index Terms*—Autonomous driving, message synchronization, ROS, sensor fusion.**

I. Introduction

MULTISENSOR data fusion is an integral component in the functionality of advanced autonomous systems like self-driving vehicles, robots, and unmanned aerial vehicles. This technology equips these systems with the capability to accurately detect and interpret their environment. Nonetheless, one practical challenge is that sensor data from various sources often lack synchronized sampling times and may face variable delays on their way to the fusion process [1], [2]. This introduces significant *time disparity*, i.e., discrepancies in the actual timing of data acquisition among the data from different sensor sources. Such time disparity may be substantial enough to compromise the integrity of the fusion outcomes, rendering the data less reliable or, in extreme cases, rendering it nonsensical. To circumvent this, it is critical to synchronize the data streams from all sensors to manage and minimize these disparities before the data proceeds to the fusion process.

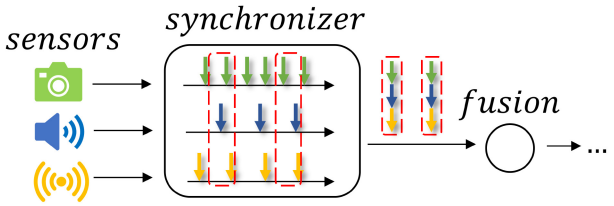

Fig. 1. Role of message synchronizer in ROS. Down arrows are messages ordered by timestamp.

The robot operating system (ROS) stands at the forefront of software development for robotics and autonomous systems. ROS provides essential mechanisms for synchronizing messages to mitigate time disparity in sensor fusion processes. Fig. 1 depicts the role of synchronizer in a typical ROS system. It subscribes the messages from sensors, choosing sets of messages from them, and publishes these sets to following components like data fusion. Significant research has already been conducted on the *ApproximateTime* message synchronization policy, a staple in ROS since version 1.1 and widely used in various ROS-based applications. These studies have rigorously defined the *ApproximateTime* policy's timing behaviors, establishing reliable upper bounds for its worst-case time disparity and latency due to the extra waiting time incurred by the synchronization. These findings have been pivotal for integrating the *ApproximateTime* policy into real-time systems design, thereby ensuring high-quality fusion output and predictable end-to-end latency for sensor data processing.

However, with the rollout of ROS's newest stable release, ROS 2 Iron Irwini, a novel *LatestTime* message synchronization policy has been introduced. Existing research does not cover this policy, leaving its timing behavior and real-time performance characteristics underexplored. This article aims to bridge this gap by delivering a comprehensive analysis of the *LatestTime* policy's timing behavior and evaluating its real-time performance, including both the worst-case time disparity and the worst-case latency a message may experience due to the synchronization with other messages.

First, our study reveals that the *LatestTime* policy has a defect in that the policy may suffer very large latency in publishing subsequent outputs, and the latency can be even

Manuscript received 11 August 2024; accepted 12 August 2024. Date of current version 6 November 2024. The work of Chenhao Wu and Naijun Zhan was supported in part by the National Key Research and Development Program of China under Grant 2022YFA1005101, and in part by NSFC under Grant 62192732. The work of Ruoxiang Li and Nan Guan was supported in part by the Hong Kong GRF under Grant 15206221 and Grant 11208522. This article was recommended by Associate Editor S. Dailey. *(Corresponding author: Nan Guan.)*

Chenhao Wu is with SKLCS, Institution of Software, Chinese Academy of Sciences, Beijing 100045, China, and also with the School of Computer Science, University of CAS, Beijing 100049, China.

Ruoxiang Li and Nan Guan are with the Department of Computer Science, City University of Hong Kong, Hong Kong, SAR (e-mail: nanguan@cityu.edu.hk).

Naijun Zhan is with the School of Computer Science, Peking University, Beijing 100871, China, also with SKLCS, Institution of Software, Chinese Academy of Sciences, Beijing 100045, China, and also with the School of Computer Science, University of CAS, Beijing 100049, China.

Digital Object Identifier 10.1109/TCAD.2024.3446709

infinite in the worst case. Consequently, the system using the *LatestTime* policy for message synchronization may suffer large and even infinite latency to obtain information update, which is unacceptable to real-time systems that have strict requirements on the information update latency. We propose a small revision to the original *LatestTime* policy to address this defect so that the latency of interest can be well bounded, and implemented in ROS 2 Iron Irwini [3].

After repairing the above-mentioned defect, we derive *safe* and *tight* upper bounds on its worst-case *time disparity* and two types of latency incurred due to message synchronization, the *passing latency* and *reaction latency* [1]. These bounds are useful for verifying timing and functional properties for real-time systems. E.g., for any downstream component that relies on synchronizer's outputs, such as data fusion, as long as our bounds fall within an acceptable range, the system can deliver the expected quality under all circumstances. Moreover, due to the closed-form nature of our bounds on system parameters, designers can easily adjust these parameters to meet given requirements, or identify the bottleneck parameters that contributes most to corresponding metrics.

We conduct experiments under variable settings to validate our conclusions, and evaluate the precision, safety and robustness of our bounds.

II. Background and Related Work

A. Message Synchronization in ROS

Currently, four standard synchronization policies are provided in the Latest version of ROS 2 — The *ExactTime* policy [4], the *ApproximateEpsilonTime* policy [5], the *ApproximateTime* policy [6], and the *LatestTime* policy.

The *ExactTime* policy selects output messages with exactly the same timestamp. Its output sets are perfectly aligned, which is ideal for data fusion. However, it is too restrictive for real-world systems and is seldom to use.

The *ApproximateEpsilonTime* policy is a generalized version of the *ExactTime* policy. It selects message set whose messages' maximal timestamp difference is no larger than a user-predefined constant ϵ, as opposed to 0 in the *ExactTime* policy. Its performance highly relies on the selection of ϵ.

The *ApproximateTime* policy predicts the incoming messages. It online selects output set with the minimal timestamp difference from all available messages, making it adaptive to incoming messages. Nonetheless, the accuracy of its prediction relies heavily on user input. Since it only predicts one incoming message, its output is a local optimum and may lead to long-term worse cases [7].

The *LatestTime* policy prioritizes output frequency, aiming to output messages at the highest frequency among sensors. It achieves this by up-sampling slower sensors using a zero-order hold, enabling frequent outputs as if all sensors operated at the highest frequency. While enhancing the system's perception of the environment, the policy does not guarantee temporal alignment in its outputs.

B. Related Works

Data fusion algorithms play a crucial role in environmental perception and decision-making processes. However, many of these algorithms assume aligned timestamps, a condition rarely met in real-world systems. Various techniques [8], [9], [10], [11] have been developed to mitigate the impact of misalignment, but their effectiveness is limited within certain thresholds of temporal inconsistency. Thus, attention to message synchronization is crucial to improve the quality of data fusion. Prior studies [12], [13], [14], [15] addressed the precise timestamping of sampled data. Thus, this article, focus on managing data from different sensors.

As the successor of ROS, ROS 2 claims substantial enhancement of real-time performance. However, hard real-time guarantees remain elusive due to dependencies on underlying middle-wares and hardware platforms [16]. Many works have evaluated its real-time performance using measurement-based approach. Maruyama et al. [17] evaluated the capabilities and performance for ROS1 and ROS2 with different DDS implementations, considering metrics like latency and throughput. Gutiérrez et al. [18] conducted communication evaluation for ROS2 real-time applications taking into account the worst-case latency. Teper et al. [19] 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. Other works aim to improve the real-time capabilities of ROS 2. Abdullah et al. [20], Randolph et al. [21], Sobhani et al. [22], and Choi et al. [23] proposed techniques to address the limitations of the default scheduling strategy of ROS2 by enhancing or redesigning the executor. However, all the above works focuses on the executor in ROS without considering the message synchronizer.

Recently, [2] conducted the first analysis of ROS synchronizer. By modeling the *ApproximateTime* policy, they proposed an upper bound on the worst-case time disparity of its output. Li et al. [24] summarized and formally proved important properties of this policy. Li et al. [1] evaluated the latency caused by the synchronization process and derived corresponding upper bounds. Instead of analyzing existing policies in ROS, [7] proposed a novel policy named "SEAM" and demonstrated its optimality regarding time disparity. The *LatestTime* is a newly introduced standard synchronization policy in ROS. We, for the first time, formally model its behavior and analyze its timing properties.

III. Problem Definition

A. System Model

The system has a total of N sensors, indexed by $1, \ldots, N$, each sporadically sampling messages and transferring them through corresponding channels. We use m_i^j to denote the jth message sampled by the ith sensor according to the temporal order of sampling. For simplicity, we may omit the superscript j when it is clear or irrelevant in the context. Each message has a timestamp $\tau(m_i^j)$, indicating the time when sensor samples it. We assume that for each sensor i, the timestamp difference between any two consecutive messages is bounded within $[T_i^B, T_i^W]$, i.e., $T_i^B \leq \tau(m_i^{x+1}) - \tau(m_i^x) \leq T_i^W$. It holds that $0 < T_i^B \leq T_i^W$ for $i \in \{1, \ldots, N\}$. $T_i^B = T_i^W$ corresponds to the periodic case.

After some preprocessing, messages are transmitted to the *Message Synchronizer*, which is the focus of this article. The

arrival time, denoted as $\alpha(m_i^j)$ refers to the time when message m_i^j arrives at the synchronizer. $\alpha(m_i^j) - \tau(m_i^j)$ is the delay caused by preprocessing and data transfer. We assume that for each sensor i, the delay of its messages are bounded within $[D_i^B, D_i^W]$. We have $0 \leq D_i^B \leq \alpha(m_i^j) - \tau(m_i^j) \leq D_i^W$. It is noteworthy that the scenario of no delay, i.e., $0 = D_i^B = D_i^W$, is also considered. We also assume that the delay does not distort the sampling order of the messages, i.e., $\tau(m_i^j) < \tau(m_k^l) \implies \alpha(m_i^j) < \alpha(m_k^l)$. Many techniques, e.g., [16], [22], and [23] have been well developed to bound the response time of real-time tasks. So we assume D_i^B and D_i^W are known values.

The *Message Synchronizer* is a software component responsible for selecting and publishing the output messages to the subsequent processing component. It invokes the algorithm called *synchronization policy* each time a new message arrives. This article focuses on the behaviors and properties of the *LatestTime* synchronization policy in ROS, which will be elaborated in Section IV. We omit the execution time of the synchronizer, and we assume that the processing of the *Message Synchronizer* is reliable, ensuring no concerns of message overflow or loss. We use the term *publish* to refer to the synchronizer outputting a set of messages. All messages in a set are published together, the time of which is referred as the *publishing time*.

B. Problem Definition

We assume the following parameters related to the input messages of the system.

1) *N:* Total number of the sensors.
2) T_i^B *and* T_i^W*:* The minimal and maximal timestamp difference between any two consecutive messages of each sensor i, respectively.
3) D_i^B *and* D_i^W*:* The minimal and maximal delay of messages in each channel i, respectively.

Based on these information, our objective is to derive upper bounds for the time disparity of the published sets, the passing latency and reaction latency of the system, whose definitions are provided below.

Definition 1 (Time Disparity [2]): Let $S = \{m_1, \ldots, m_N\}$ be a set of messages , where m_i is sampled by sensor i. The time disparity of S, denoted by $\Delta(S)$, is defined as the difference between the largest and smallest timestamps of messages in S, i.e.,

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

Definition 2 (Passing Latency [1]): Let m_i^j be a message in a published message set which is published at time t_f. The passing latency of m_i^j, denoted as $P(m_i^j)$, is defined as the difference between its publishing time and its arrival time, i.e.,

$$P\left(m_i^j\right) = t_f - \alpha\left(m_i^j\right).$$

Definition 3 (Reaction Latency [1]): Let m_i^k be a message, t_f denote the time when m_i^k is first published, and m_i^j be the latest published message in $\{m_i^l \mid l < k\}$. The reaction latency experienced by m_i^k, denoted as $R(m_i^k)$, is defined as the difference between the arrival time of m_i^j and the first publishing time of m_i^k

$$R\left(m_i^k\right) = t_f - \alpha\left(m_i^j\right).$$

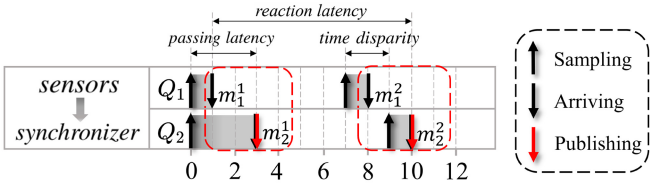

Fig. 2. Illustration of three timing metrics.

Fig. 2 illustrates these metrics. In this example, there are 2 sensors, each sampling messages and transmitting them to synchronizer via a corresponding channel. The x-axis represents time. The sampling and arriving time of each message are denoted as upward and downward pointing arrows, respectively. The shaded area between them indicates the delay. The synchronizer selects messages from each channel and publish the whole message set, circled by red box, at publishing time depicted as red arrow.

The time disparity of the second published set is depicted in the figure, which equals to the difference of the sampling times of m_1^2 and m_2^2. The time disparity represents the sampling time inconsistency of the published messages. The larger the time disparity, the lower the quality of data fusion's input.

The passing latency of m_1^1 is equal to the difference of the arriving time of m_1^1 and its publishing time (red arrow near m_2^1). An important system-level timing metric is the end-to-end delay of a message, which is the interval from its sampling to when the actuator takes action based on it. Passing latency measures the contribution introduced by the synchronizer to the end-to-end delay.

The reaction latency corresponding to m_1^2 is the difference of the arrival of the last published message before it in this channel (m_1^1), and its first publishing time (red arrow of m_2^2). The reaction latency is the maximum delay introduced by synchronizer as a part of the end-to-end reaction time, which is the time the whole system needs to react to an external event regarding a specific sensor. In a worst-case scenario, an external event happens just after m_1^1 is sampled. So $sensor_1$ does not detect it until m_1^2 is sampled. Thus, the latency caused by synchronizer regarding reacting to this event is exactly the reaction latency of m_1^2.

In this article, we focus on finding safe upper bounds of the three metrics, depending only on the aforementioned information. By safe we mean that the bounds always hold, irrespective of the specific message sampling pattern, the actual message delay, and any other runtime-determined values. Given that the bounds are safe, we strive for their tightness, which refers that there exists a specific system, the worst-case value of whose corresponding metric matches or approaches the value given by the bound.

IV. *LatestTime* POLICY

The *LatestTime* policy is one of the standard message synchronization policies in ROS [25]. It is newly introduced in ROS2 C++ version *Iron Irwini* and *Rolling Ridley*, yet no research has undertaken an analysis of its behaviors and timing properties. In the following, we derive a model of the policy to assess its timing behavior, focusing on the time disparity, passing latency, and reaction latency. We also present an illustrative example for better understanding.

The *LatestTime* policy aims to publish at the highest frequency among all channels. Messages from the slower sensors will be repeated at this frequency and they are updated whenever a new one arrives. Therefore, from the signal processing perspective, this is effectively upsampling the slower messages using a zero-order hold.

The *LatestTime* policy uses a message set S to maintains the newest message of each sensor. We use $S[i]$ to denote the ith message in S, which is the newest from sensor i.

The synchronizer is only invoked upon the arrival of a new message. Each time invoked, it utilizes two key components, *data update* process and *pivot selection* process, to achieve its goal.

A. Data Update

The *LatestTime* policy statistically estimate the frequency of each channel. The frequency corresponding to two consecutive messages in channel i is denoted as $f(m_i^x, m_i^{x+1}) = [1/\alpha(m_i^{x+1}) - \alpha(m_i^x)]$, which is the reciprocal of the difference between the messages' arrival times. The mean value of the frequency is denoted as $\bar{f}_i$, representing the estimated frequency of a channel and is used for the further comparison. The policy updates $\bar{f}_i$ online based on frequency of all the arrived messages in channel i. The calculation method it adopts is a modified version of the exponential moving average (EMA) method. Definition of EMA and the implementation of *data update* is detailed in Appendixes A and B.

Due to jitters in message arrival times, the frequency fluctuates around its mean value. To estimate this variation, the system calculates the mean error for each channel i, denoted as $\bar{e}_i$. The calculation details are provided in Appendix B. By counting the mean error, the system wants to estimate the range of frequency. Given an user-predefined constant γ_i, referred to as the margin factor, the system estimates the maximal frequency variation as $\gamma_i * \bar{e}_i$. Thus, the estimated frequency range is $[\bar{f}_i - \gamma_i \cdot \bar{e}_i, \bar{f}_i + \gamma_i \cdot \bar{e}_i]$. If a new frequency falls outside this range, the system considers the mean frequency unsafe.

B. Pivot Selection

Based on the statistics, the system aims to publish at the highest frequency among channels. Since the system is triggered upon the arrival of messages, this task simplifies to deciding whether to publish upon the arrival of each message. Next we introduce the other key component, *pivot selection*, which accomplishes this task.

The *pivot selection* involves more than simply choosing the most frequent channel. It also checks if the mean frequency is safe at the current moment. For each channel i, if the current time has exceeded the threshold given by statistical lower bound of frequency $\bar{f}_i - \gamma_i \cdot \bar{e}_i$, the next frequency will be even lower, rendering the mean frequency unsafe. Theoretically the system could obtain this information as soon as the time surpasses this threshold. However, since the *data update* of a channel only proceeds upon the arrival of a new message, the system will not correct its estimation until next message arrives. Therefore, the *pivot selection* has to check all the mean frequencies to exclude those unsafe.

Algorithm 1: Pivot Selection

Input: the new message m_i
Data: the message set S, candidate set C, statistics of each channel j: $\bar{f}_j$ and $\bar{e}_j$, margin factor of each channel j: γ_j
Output: the pivot p

```
1  C ← {i};
2  for j ← 1 to N do
3      if ē_j has not been initialized yet then
4          C.add(j);
5      else
6          h ← f(S[j], m_i);
7          if h ≥ f̄_j − γ_j ∗ ē_j then
8              C.add(j);
9  p ← element in C with maximal f̄;
10 return p;
```

Algorithm 1 shows the pseudo code of *pivot selection*. The system selects a candidate set C for the pivot, encompassing indexes of channels with safe mean frequency. Since *data update* works before *pivot selection* and the mean frequency of the arriving message's channel is just updated, it is added to C (line 1). For channels with too few arrived messages to calculate $\bar{e}$, the system regards them as safe candidates (lines 3 and 4). Then, for each channel j, the system computes a frequency h using current time $\alpha(m_i)$ and previous arriving time $\alpha(S[j])$ (line 6). This frequency represents the highest possible frequency of the next message. If h falls below the statistical lower bound $h < \bar{f}_j - \gamma_j * \bar{e}_j$, the next frequency will be even lower and $\bar{f}_j$ is unsafe. By excluding such channels, the system identifies all the candidates (lines 7 and 8). Finally, the pivot is selected as the channel with the highest frequency among candidates (line 8).

C. Model

In the following we present a model of the *LatestTime* policy. This model serves as a semantics approximation of the original policy, abstracting away details that do not affect our analysis. Accordingly, we aim to capture its behaviors that dictate the three metrics we concern. The correctness of this abstract model will be assessed in Section VII.

Algorithm 2 shows the pseudo code of the *LatestTime* policy. Upon the arrival of a new message, if it is the first one in a channel, the system simply sets $S[i]$ and returns

Algorithm 2: *LatestTime* Policy

Input: the newly arrived message m_i
Data: the set of the newest messages of each channel S
1 **if** m_i *is the first message in channel* i **then**
2 $S[i] \leftarrow m_i$;
3 **return**;
4 executing *data update*;
5 $p \leftarrow$ *pivot selection*;
6 **if** S *is full and* $i == p$ **then**
7 publish S;
8 $S[i] \leftarrow m_i$;
9 **return**;

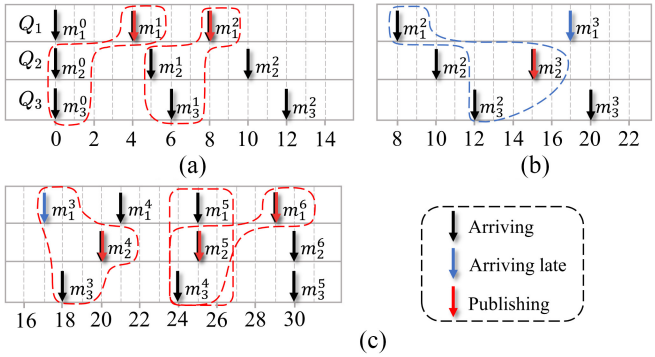

Fig. 3. Example to illustrate the *LatestTime* policy. (a)–(c) Are three sequential fragments of a time interval. x-axis is time.

(lines 2 and 3). Otherwise, the system executes *data update* to update the statistics in this channel (line 4). Then the pivot is determined by *pivot selection* (line 5). The system publishes S if the pivot coincides with the current channel (line 6).

D. Illustrative Example

Fig. 3 presents an example to illustrate the *LatestTime* policy. The system has three sensors that sample messages and transmit them through corresponding channels denoted as Q_i. Q_2, and Q_3 have fixed sampling periods of 5 and 6, respectively. The sampling period of the first channel ranges from $T_1^B = 4$ to $T_1^W = 10$. For simplicity, we assume that there are no message delays. In addition, user-predefined parameters used in *data update* are set as $\beta_i^f = 0.3$, $\beta_i^e = 0.3$, and $\gamma_i = 10$ for each channel i.

In Fig. 3(a), at time 0, the first message of each channel arrives, and the system simply puts them into the message set. At time 4, m_1^1 arrives. $\bar{f}_1$ is initialized to 0.25. As it is the only channel with valid $\bar{f}$, it is chosen as the pivot and the current message set is published (marked by the red dashed box). Subsequently, m_2^1 and m_3^1 arrive. $\bar{f}_2$ and $\bar{f}_3$ are initialized to 0.2 and 0.17, respectively. The system does not publish at these times because channel 1 has the highest frequency and serves as the pivot. Then a new wave of messages m_i^2 arrives and $\bar{f}_i$ remain unchanged. Given that channel 1 is still the pivot, the system publishes the message set, including $\{m_1^2, m_2^1, m_3^1\}$ upon the arrival of m_1^2. It is worth mentioning that during the *pivot selection* upon the arrival of m_3^2, the system assesses whether the next message of channel 1, m_1^3, will be late. Based on the statistics of channel 1: $\bar{f}_1 = 0.25$, $\bar{e}_1 = 0$, and $\gamma_1 = 10$, the system calculates the acceptable bound of $\alpha(m_1^3)$ as 12. Since the current time does not exceed the bound, the system considers $\bar{f}_1$ safe and chooses 1 as the pivot.

In Fig. 3(b), at time 15, m_2^3 arrives, while m_1^3, which is expected to arrive earlier than time 12, has not yet arrived. The *data update* calculates $\bar{f}_2 = 0.2$ and $\bar{e}_2 = 0$. During *pivot selection*, the system now determines that m_1^3 will be late and $\bar{f}_1$ is unsafe. As a result, channel 1 is excluded from candidates, and channel 2 becomes the new pivot. The message set at this time, $\{m_1^2, m_2^3, m_3^2\}$, is published (marked with blue dashed box). After that, m_1^3 arrives at 18. During *data update* of this channel, it is determined to be late and $\bar{f}_1$ is updated to 0.1, which is now the smallest among all channels.

Fig. 3(c) shows the changes after the late message m_1^3. This part serves as an example of the calculation detail of *data update* illustrated in Algorithm 4 in Appendix B. At time 20, m_2^4 arrives, with $\bar{f}_1 = 0.1, \bar{f}_2 = 0.2$ and $\bar{f}_3 = 0.16$. Consequently, channel 2 becomes the pivot and the message set $\{m_1^3, m_2^4, m_3^3\}$ is published. Subsequently, at time 21, m_1^4 arrives. The phase of channel 1 is 2. $\bar{f}_1$ is updated to $0.3 \times 0.25 + 0.7 \times 0.1 = 0.145$ and $\bar{e}_1$ is initialize. At time 25, m_1^5 and m_2^5 arrive. Assuming m_1^5 arrives slightly earlier and is processed first, *data update* of phase 3 of channel 1 updates $\bar{f}_1$ to $0.3 \times 0.25 + 0.7 \times 0.145 = 0.18$. Since 0.18 is still less than $\bar{f}_2 = 0.2$, channel 2 remains the pivot. The system publishes the message set after processing m_2^4, which includes $\{m_1^5, m_2^5, m_3^4\}$. Finally at time 29, m_1^6 arrives. This time $\bar{f}_1$ is updated to $0.3 \times 0.25 + 0.7 \times 0.18 = 0.201$, which is larger than $\bar{f}_2$. Consequently, channel 1 becomes the new pivot and the message set $\{m_1^6, m_2^5, m_3^4\}$ is published.

V. Timing Defect and Revision

In this section we report a timing defect in the current implementation of the *LatestTime* policy. This defect refers to that, even if the timestamp interval and delay of each channel are bounded, i.e., messages of each channel keep arriving, it is possible for the *LatestTime* policy to refrain from publishing for an arbitrarily long time. This defect not only obstructs the policy from achieving its goal of publishing at the highest frequency, but also leads to risks in real-world tasks, e.g., not publishing messages for a long time may cause the autonomous vehicle being unable to react to an emergency. We will present a simple modification of the policy, which repairs the defect while preserving most of its implementation and semantics. Below we first reveal the defect.

A. Timing Defect

Below we present a type of message arriving pattern that triggers the timing defect. The problem lies in the condition to publish. Algorithm 2 shows that the policy publishes only when message arrives from the pivot channel (line 6). If the pivot remains unchanged, the system will continuously publish as the messages keep arriving. However, when pivot changes, this condition may be always false. Let's consider a system whose channels have very similar frequencies. The

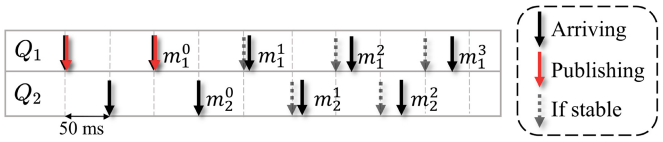

Fig. 4. Example of worst-case pattern triggering timing defect. The pattern starts from m_1^0. Each subsequent message arrives slightly later to avoid publishing. Dashed arrow is a reference of arriving stably.

mean frequency of a channel will be affected by its arriving message. Suppose δ is a small positive value, if a new message arrives δ earlier, its mean frequency will slightly increase, making its channel the pivot. Conversely, if it arrives δ later, its mean frequency will decrease slightly and its channel will not be the pivot. Therefore, we can control the system to publish or not upon arrival of each message by "slightly" advancing or delaying it. Since δ can be sufficiently small, the publishing behavior is nearly arbitrary for similar message arriving patterns. Below, we construct a worst-case example by slightly delaying every new message. We prove by calculation that the inputting channel will not match the pivot for an arbitrarily long time, leading to the timing defect.

Fig. 4 shows our worst-case example. The system consists of two sensors $N = 2$. Both sensors have $T^B = 100$ms and $T^W = 200$ms. For simplicity, the delay of both sensors is ignored, $D^W = 0$. At the beginning, both sensors sample messages at the highest frequency $1/T^B = 10$Hz. The messages' arrivals of two sensors are interleaved with a time difference of $\alpha(m_2^i) - \alpha(m_1^i) = 50$ms. The system estimates the mean frequency of both channels to be 10Hz and always selects channel 1 as the pivot. The parameters used in *data update* are $\beta_i^f = 0.9$, $\beta_i^e = 0.3$ and $\gamma_i = 10$ for $i \in \{1, 2\}$.

When m_i^0 arrives, the system publishes because pivot $= 1$. After $50ms$, m_2^0 arrives. Since the statistics and the pivot remain unchanged, the system does not publish.

After that, each message arrives slightly later to decrease its mean frequency to be $(10 - n\delta)$Hz, where δ is a small positive value and n is a gradually increasing natural number starting from 0. Based on the model of *data update* in Appendix B, we can reversely calculate the needed arriving time of each message. The calculation result is shown in Appendix C, which shows no contradiction and validates the feasibility of the defect in Fig. 4. Since δ can be arbitrarily small, this pattern can last for an arbitrarily long time before constrained by T_i^W. Therefore, the next published set can be infinitely delayed.

B. Revision

The timing defect shows that, in the scenario that the pivot changes, the publishing behavior cannot be guaranteed relying solely on the pivot. Conditions in line 6 of Algorithm 2 needs to be enhanced. In a real-world task, publishing at a higher frequency may result in redundant information and increased system workload, but is less harmful to the safety. On the contrary, publishing infrequently can lead to inability to respond to external events and compromise safety. Therefore, our enhancement focuses on preventing the publishing frequency from dropping too low. Algorithm 3 presents the pseudo code of the revised policy.

Algorithm 3: Revised *LatestTime* Policy

Input: the newly arrived message m_i
Data: the set of the newest message of each channel S, last publishing time t_l

```
1  if m_i is the first message in channel i then
2      S[i] ← m_i;
3      return;
4  data update of channel i;
5  p ← pivot selection;
6  if S is full then
7      if i == p or 1/(α(m_i) − t_l) ≤ f̄_p then
8          publish S;
9          t_l = α(m_i);
10 S[i] ← m_i;
11 return;
```

In Algorithm 3, the red text is our modification from Algorithm 2. The system maintains the last publishing time using a new datum t_l. It is updated whenever the system publishes (line 9). When current time since last publishing has exceeded the corresponding interval of the pivot's frequency, the system publishes even if the current channel is not the pivot (line 7). This enhancement guarantees that even if the pivot changes and the inputting channel fails to match the pivot, the system still publishes upon the arrival of the first message after exceeding the time interval corresponding to the pivot's frequency. Considering that when pivot does not change, the system keeps publishing whenever messages arrive from the pivot, we can infer that the revised policy can guarantee to keep publishing messages irrelevant to the inputting pattern. It now refrains from the timing defect. We will prove this property by giving a reaction latency upper bound in Section VI-C.

Our revision also introduces extra cost. In terms of space, the system needs to store t_f. In terms of time, it needs to evaluate the new condition upon messages' arrival. Considering the storage of all the statistics and the massive calculation in the *data update* process, the cost we incur is negligible.

VI. Upper Bound Analysis

In this section, based on the revised policy, we derive safe and tight upper bounds for the time disparity, passing latency and reaction latency. Safety is guaranteed by formal proofs and we present examples as evidence of the tightness. Below we use notation $A_i = T_i^W + D_i^W - D_i^B$ to ease our expression.

A. Time Disparity Upper Bound

In the following we derive an upper bound for the time disparity of any published set of the LatestTime policy. According to Algorithm 3, the published set is always the message set S maintained by the system. Hence, our analysis focuses on the time disparity of the message set S. We define

the lifespan of a message in S as the time interval from when it is added to S until it is replaced. To begin, we establish the upper bound of the lifespan of a message.

Lemma 1: Let m_i^j be a message in channel i, and $l(m_i^j)$ denote the lifespan of m_i^j. Let T_i^W be the maximal sampling interval of sensor i, D_i^B and D_i^W be the minimal and maximal delay of channel i, respectively. we have

$$l\left(m_i^j\right) \leq T_i^W + D_i^W - D_i^B.$$

Proof: According to Algorithm 3, the message set is updated upon the arrival of a new message. Lines 2 and 10 replaces the message of the same channel in S with the new message. Consequently, each message is added to S upon its arrival and is replaced upon the arrival of the next message in the same channel. Therefore, we have

$$l\left(m_i^j\right) = \alpha\left(m_i^{j+1}\right) - \alpha\left(m_i^j\right) \leq T_i^W + D_i^W - D_i^B.$$

■

Since the time disparity focuses on the timestamp rather than the arrival time, we derive another upper bound of lifespan that incorporates timestamp on one side. This difference enables us to obtain a tight upper bound of time disparity.

Lemma 2: Let m_i^j denote a message in the ith channel. We denote the end time of the lifespan as $l_e(m_i^j)$. Then we have

$$l_e\left(m_i^j\right) - \tau\left(m_i^j\right) \leq T_i^W + D_i^W.$$

Proof: According to Algorithm 3, a message's lifespan ends upon the arrival of the next message in the same channel. So we have

$$l_e\left(m_i^j\right) - \tau\left(m_i^j\right) = \alpha\left(m_i^{j+1}\right) - \tau\left(m_i^j\right) \leq T_i^W + D_i^W.$$

■

Based on Lemma 2, we can establish an time disparity upper bound of any message set, and consequently, the published set.

Lemma 3: The time disparity of a message set S is upper-bounded by

$$\overline{\Delta} = \max_{i\in\{1,\ldots,N\}}\left(T_i^W + D_i^W\right) - \min_{i\in\{1,\ldots,N\}} D_i^B.$$

Proof: Let m_i represent the message with the latest timestamp in S, i.e., $\tau(m_i) = \max_{m\in S}\tau(m)$, and m_j the earliest $\tau(m_j) = \min_{m\in S}\tau(m)$. According to the assumption that delays do not distort the order of timestamps, we know that m_i arrives later than m_j. Since the arrival time of a message coincides with the start time of its lifespan, denoting the start time of the lifespan by $l_s(\cdot)$, we have

$$l_s\left(m_j\right) \leq l_s(m_i). \tag{1}$$

If S contains two messages at a time, their respective lifespans are bounded to intersect. Based on (1), if the lifespans of channels i and j intersect, the end time of m_j's lifespan must be later than the start time of m_i's. Denoting the end time of the lifespan by $l_e(\cdot)$, we have

$$l_s(m_i) \leq l_e\left(m_j\right). \tag{2}$$

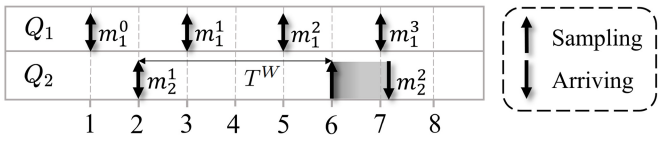

Fig. 5. Worst-case example of time disparity and passing latency.

Based on (2) and Lemma 2, we have

$$\begin{aligned}\Delta(S) &= \max_{m\in S}\tau(m) - \min_{m\in S}\tau(m)\\ &= \tau(m_i) - \tau\left(m_j\right)\\ &\leq \alpha(m_i) - \tau\left(m_j\right) - D_i^B\\ &= l_s(m_i) - \tau\left(m_j\right) - D_i^B\\ &\leq l_e\left(m_j\right) - \tau\left(m_j\right) - D_i^B\\ &\leq T_j^W + D_j^W - D_i^B\\ &\leq \max_{i\in\{1,\ldots,N\}}\left(T_i^W + D_i^W\right) - \min_{i\in\{1,\ldots,N\}} D_i^B.\end{aligned}$$

■

Theorem 1: Denote the maximal timestamp interval of channel i as T_i^W, delay range of channel i as $[D_i^B, D_i^W]$. The time disparity of a published set $S^{\mathbf{PUB}}$ is upper bounded by

$$\overline{\Delta} = \max_{i\in\{1,\ldots,N\}}\left(T_i^W + D_i^W\right) - \min_{i\in\{1,\ldots,N\}} D_i^B.$$

Proof: According to Algorithm 3 line 8, the published set is the message set S. Based on Lemma 3, the theorem holds.

■

Next we use the example in Fig. 5 to illustrate the tightness of Theorem 1. In this system, there are two channels, labeled Q_1 and Q_2, respectively. Channel Q_1 has a fixed and shorter timestamp interval $T_1^B = T_1^W = 2$, while channel Q_2 has a fixed and longer time interval $T_2^B = T_2^W = 4$. The delay bounds for these channels are $D_1^B = D_1^W = D_2^B = 0$, and $D_2^W = 1 + \delta$, where δ is a small positive value. The system has been running for some time, allowing it to acquire statistics for each channel. We assume $\bar{f}_1 < \bar{f}_2$ at time 1.

Consider m_2^1 whose timestamp is 2. Upon the arrival of m_1^1, m_1^2, and m_1^3, the system selects channel 1 as the pivot and publishes the message set including m_2^1. Note that although m_1^3 has a later timestamp than m_2^2, it arrives without delay while m_2^2 arrives after the maximum delay $1 + \delta$ (represented as shaded area). Therefore, m_1^3 arrives δ earlier. Upon its arrival, m_2^2 has not yet arrived and m_2^1 is still in the message set.

We focus on m_2^1 and the message set $S^{\mathbf{PUB}}$ published upon the arrival of m_1^3. The actual time disparity of $S^{\mathbf{PUB}}$ is

$$\Delta\left(S^{\mathbf{PUB}}\right) = \tau\left(m_1^3\right) - \tau\left(m_2^1\right) = 5.$$

The time disparity upper bound given by Theorem 1 is

$$\begin{aligned}\overline{\Delta} &= \max_{i\in\{1,\ldots,N\}}\left(T_i^W + D_i^W\right) - \min_{i\in\{1,\ldots,N\}} D_i^B\\ &= T_2^W + D_2^W - D_1^B = 5 + \delta.\end{aligned}$$

Thus, the overestimation of the time disparity upper bound in this worst-case scenario approaches 1 as δ approaches 0, showing the tightness of Theorem 1.

It is important to mention that although the above analysis is based on the revised policy in Algorithm 3, it equally applies to the original policy in Algorithm 2 since our modification has no impact on its correctness. Theorem 1 is a safe and tight time disparity upper bound of the original policy as well.

B. Passing Latency Upper Bound

The passing latency of a message refers to the time interval from its arrival to its publishing. Algorithm 3 tells us the published set is the message set S. Thus, a message can only be published within its lifespan. Based on the bound of lifespan, we can derive an upper bound on the passing latency.

Theorem 2: Denote the maximal sampling interval of channel i as T_i^W, delay range of channel i as $[D_i^B, D_i^W]$. Denote $T_i^W + D_i^W - D_i^B$ as A_i. For the published set $S^{\mathbf{PUB}}$, the passing latency of $m_i \in S^{\mathbf{PUB}}$ is upper bounded by

$$\overline{P} = A_i.$$

Proof: We denote the start time and the end time of the lifespan of m_i as $l_s(m_i)$ and $l_e(m_i)$, respectively, and the publishing time of $S^{\mathbf{PUB}}$ as t_f. According to Algorithm 3 line 8, the published set is S. Thus, t_f must fall within the lifespan of m_i. Based on Lemma 1, we have

$$P(m_i) = t_f - \alpha(m_i) = t_f - l_s(m_i) \le l_e(m_i) - l_s(m_i) \le A_i.$$

■

Example in Fig. 5 is also a worst-case scenario for the passing latency, showing the tightness of Theorem 2. We still focus on m_2^1 and message set $S^{\mathbf{PUB}}$ published upon the arrival of m_1^3. The actual passing latency of m_2^1 respecting $S^{\mathbf{PUB}}$ is

$$P\left(m_2^1\right) = \alpha\left(m_1^3\right) - \alpha\left(m_2^1\right) = 5.$$

The passing latency upper bound given by Theorem 2 is

$$\overline{P} = \max_{i\in\{1,\ldots,N\}} A_i = A_2 = 5 + \delta.$$

Thus, as δ approaches 0, the passing latency upper bound can be approached, indicating Theorem 2 is tight.

Similar to the time disparity upper bound, safety and tightness of the passing latency upper bound is not affected by our revision. Theorem 2 applies equally to the original policy.

C. Reaction Latency Upper Bound

In the following we derive an reaction latency upper bound of the revised *LatestTime* policy. Recall the reaction latency of a message m_i^k is the difference between the arrival time of the latest published message m_i^j, where $j < k$, and the first publishing time of m_i^k, t_f^k. We split this interval into two parts: 1) the latency from the arrival of m_i^j to its last publishing time t_f^j and 2) the latency from t_f^j to t_f^k. Note that no message m_i^l, where $j < l < k$, is published by definition. Since t_f^j and t_f^k are the last and first publishing time of m_i^j and m_i^k, respectively, t_f^j and t_f^k are two consecutive publishing times. The first part is the passing latency of m_i^j and its upper bound is given by Theorem 2. Below we bound the second part.

Algorithm 4: Data Update

Input: the new message m_i
Data: set of the newest messages of each channel S
1 $f \leftarrow \frac{1}{\alpha(m_i)-\alpha(S[i])}$;
2 $e \leftarrow |f - \bar{f}_i|$;
3 **if** $\Theta_i = 1$ **then**
4 $\quad \bar{f}_i \leftarrow f$, $\Theta_i \leftarrow 2$;
5 $\quad$ **return**;
6 **if** $\Theta_i = 2$ **then**
7 $\quad \bar{f}_i \leftarrow \beta_i^f \times f + (1 - \beta_i^f) \times \bar{f}_i$;
8 $\quad \bar{e}_i \leftarrow e$, $\Theta_i \leftarrow 3$;
9 $\quad$ **return**;
10 **if** $\Theta_i = 3$ **then**
11 $\quad$ **if** $e <= \gamma_i \cdot \bar{e}_i$ **then**
12 $\quad\quad \bar{f}_i \leftarrow \beta_i^f \times f + (1 - \beta_i^f) \times \bar{f}_i$;
13 $\quad\quad \bar{e}_i \leftarrow \beta_i^e \times e + (1 - \beta_i^e) \times \bar{e}_i$;
14 $\quad$ **else**
15 $\quad\quad \bar{f}_i \leftarrow f$, $\Theta_i \leftarrow 2$;
16 $\quad$ **return**;
17 **return**;

Lemma 4: For channel i, within any time interval of length A_i, $[t, t + A_i]$, at least one message arrives.

Proof: We prove by contradiction. Suppose no message arrives within $[t_0, t_0 + A_i]$. let m_i^x be the message arrives before t_0 and m_i^{x+1} the one arrives after $t_0 + A_i$. We have

$$\begin{aligned} T_i^W &\ge \tau\left(m_i^{x+1}\right) - \tau\left(m_i^x\right) \\ &= \alpha\left(m_i^{x+1}\right) - \alpha\left(m_i^x\right) + D_i^x - D_i^{x+1} \\ &> T_i^W + D_i^W - D_i^{x+1} + D_i^x - D_i^B \\ &\ge T_i^W. \end{aligned}$$

This contradiction implies that the assumption does not hold and the lemma is proved. ■

Lemma 5: For a channel i, its mean frequency $\bar{f}_i$ is lower bounded by

$$\bar{f}_i \ge \frac{1}{A_i}.$$

Proof: We need to analyze the detail of the *data update* in Appendix B. In Algorithm 4, line 11 implies that f is the frequency corresponding to the time interval $\alpha(m_i) - \alpha(S[i])$, where m_i is the arriving message and $S[i]$ is the preceding message in this channel. Thus, we have

$$f = \frac{1}{\alpha(m_i) - \alpha(S[i])} \ge \frac{1}{A_i}. \tag{3}$$

Next we prove by induction. There are totally four program points that modify $\bar{f}_i$: lines 4, 7, 12, and 15. In the following we use $\bar{f}_i^x$ to distinguish different value of $\bar{f}_i$ after being updated by these lines. If updated by Lines 4 or 15, $x = 0$.

If updated by lines 7 or 12, x increases by 1. Lines 4 and 15 assign f to $\bar{f}_i$. Based on (3) we have

$$\bar{f}_i^0 = f \geq \frac{1}{A_i}. \tag{4}$$

Lines 7 and 12 update $\bar{f}$

$$\bar{f}_i^{x+1} = \beta_i^f \times f + \left(1 - \beta_i^f\right) \times \bar{f}_i^x.$$

Given that $0 \leq \beta_i^f \leq 1$, we know $\bar{f}_i^{x+1}$ is a linear combination of $\bar{f}_i^x$ and f. Based on induction hypothesis $\bar{f}_i^x \geq (1/A_i)$, we have

$$\bar{f}_i^{x+1} \geq \min\left(\bar{f}_i^x, f\right) \geq \frac{1}{A_i}. \tag{5}$$

Consequently, based on (4) and (5), we have

$$\forall x, \bar{f}_i^x \geq \frac{1}{A_i}.$$

Thus, the lemma always holds. ■

Based on Lemmas 4 and 5, we derive the upper bound of two consecutive publishing times.

Lemma 6: Let $S_1^{\mathbf{PUB}}$ and $S_2^{\mathbf{PUB}}$ be two consecutive published sets. Denoting the publishing times of them as t_f^1 and t_f^2, respectively, where $t_f^2 > t_f^1$. The following inequality holds:

$$t_f^2 - t_f^1 \leq 2 \min_{i \in \{1,\ldots,N\}} A_i.$$

Proof: For a channel i, let's consider the time interval $[t_f^1 + A_i, t_f^1 + 2A_i]$. Based on Lemma 4, there is at least one message in channel i arriving within this interval. We denote this message as m_i.

If the system publishes before the arrival of m_i, we have

$$t_f^2 \leq \alpha(m_i) \leq t_f^1 + 2A_i.$$

So we have

$$t_f^2 - t_f^1 \leq 2A_i. \tag{6}$$

Next we prove that, if the system has not published yet, it must publish upon the arrival of m_i, so that (6) also holds.

Upon the arrival of m_i, the *data update* will update the mean frequency $\bar{f}_i$. Based on Lemma 5, we have the lower bound of the updated mean frequency

$$\bar{f}_i \geq \frac{1}{A_i}. \tag{7}$$

Based on the lower bound of $\alpha(m_i)$, we have

$$\alpha(m_i) - t_f^1 \geq A_i. \tag{8}$$

Combining (7) and (8), we have

$$\bar{f}_i \geq \frac{1}{\alpha(m_i) - t_f^1}.$$

Since no publishing happens within $(t_f^1, \alpha(m_i))$, we have $t_l = t_f^1$ at this time. Denoting the pivot as p, we have

$$\bar{f}_p \geq \bar{f}_i \geq \frac{1}{\alpha(m_i) - t_l}.$$

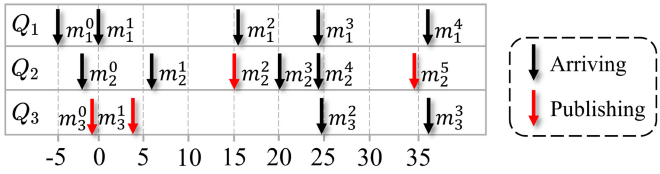

Fig. 6. Worst-case example of reaction latency.

Consequently, upon the arrival of m_i, the condition in line 7 of Algorithm 3 must be true. The system will publish at this time. Thus, in this case

$$t_f^2 - t_f^1 = \alpha(m_i) - t_f^1 \leq 2A_i.$$

Therefore (6) always holds for any given channel i. Specially, it holds for the channel with $\min A_i$ and the lemma is proved. ■

Theorem 2 and Lemma 6 bound the two parts of the reaction latency, respectively. Combining them, we derive an reaction latency upper bound of the revised *LatestTime* policy.

Theorem 3: Let $A_i = T_i^W + D_i^W - D_i^B$, where T_i^W is sensor i's maximal sampling interval, $[D_i^B, D_i^W]$ is channel i's delay range, the reaction latency of m_i is bounded by

$$\overline{R} = A_i + 2 \min_{j \in \{1,\ldots,N\}} A_j.$$

Proof: Suppose m_i' is the last published message before m_i in the same channel. Let t_f^1 be the last publishing time of m_i', t_f^2 be the first publishing time of m_i. By Theorem 2, we have

$$t_f^1 - \alpha\left(m_i'\right) \leq A_i. \tag{9}$$

From Lemma 6, we have

$$t_f^2 - t_f^1 \leq 2 \min_{i \in \{1,\ldots,N\}} A_i. \tag{10}$$

From (9) and (10), the theorem holds. ■

Below we use the example in Fig. 6 to demonstrate the tightness of the bound given in Theorem 3. There are three channels, denoted as Q_i, $i = 1, 2, 3$. Timestamp difference bounds of each channel are $T_1^B = T_2^B = T_3^B = 0$, $T_1^W = 15$, $T_2^W = 9$ and $T_3^W = 50$. Delay bounds of each channel are $D_1^B = D_2^B = D_3^B = 0$, $D_1^W = \delta$, and $D_2^W = D_3^W = 1$, where δ is a small positive value. The parameters used in *data update* are as follows: the EMA weights of frequency are $\beta_1^f = \beta_2^f = 0$ and $\beta_3^f = 1$, resulting in $\bar{f}_1$ and $\bar{f}_2$ fixed. Margin factors are large enough, i.e., $\gamma_i \approx \infty, i = 1, 2, 3$, eliminating the need to consider the reboot process in phase 3 of *data update*. Therefore, $\beta_1^e, \beta_2^e, \beta_3^e$ are irrelevant. Note that these extreme configurations is to simplify our calculation below and the worst-case example does not rely on this extremeness.

The system has been running for a while before time 0. We assume that the interval between the arriving times of the first and second messages of channel 1 (resp. 2) is 13 in (10). So $\bar{f}_1$ and $\bar{f}_2$ are fixed as $\bar{f}_1 = 1/13$ and $\bar{f}_2 = 1/10$. We assume channel 3 has the highest frequency among all channels before time 0. At time -1, m_3^0 arrives and the system publishes.

m_1^1 arrives with the lowest delay at time 0. At time 4, m_3^1 arrives. The system selects channel 3 as pivot and publishes. At time 6 and 15, m_2^1 and m_2^2 arrive sequentially. Since $\bar{f}_2$ is

fixed as $(1/10) < (1/5) = \bar{f}_3$, the system does not publish upon the arrival of m_2^1. But it publishes upon the arrival of m_2^2 since $(1/[\alpha(m_2^2) - t_l]) = (1/11) < (1/5) = \bar{f}_p$, and the added condition is satisfied.

At time $15 + \delta$, m_1^2 arrives after the longest arriving time interval from m_1^1. m_2^3 arrives at time 20. At time $25-2\delta$, m_1^3 and m_2^4 arrive. We assume m_2^4 arrives with the lowest delay. The system does not publish upon these messages' arrival because the pivot is channel 3, and the second part the condition in Algorithm 3 line 7 is true only when the time passes 25. Finally at time $35 - 2\delta$, m_2^5 arrives with the longest delay $D_2^W = 1$. Its arrival time exceeds 25, so the condition is satisfied and the system publishes.

We focus on the reaction latency of m_1^3. The preceding published message from sensor 1 is m_1^1. We have $\alpha(m_1^1) = 0$ and $t_f^2 = \alpha(m_2^5) = 35 - 2\delta$. The actual reaction latency is

$$t_f^2 - \alpha\left(m_1^1\right) = 35 - 2\delta.$$

The reaction latency upper bound given by Theorem 3 is

$$\bar{R} = A_i + 2 \min_{j\in\{1,\ldots,N\}} A_j = A_1 + 2A_2 = 35 + \delta.$$

Since δ can be arbitrarily small, the overestimation of Theorem 3 approaches 1, and thus our bound is tight.

Next we explain the meaning of our reaction latency upper bound. Lemma 6 is not so intuitive by constructively considering a message in time interval $[t_f^1 + A_i, t_f^1 + 2A_i]$. We first show the insight behind it. We use T_p to denote the length of the interval corresponding to $\bar{f}_p$. The original policy guarantees that, in $[t_l, t_l + T_p]$, the system publishes as soon as receiving a message from pivot. The condition we add in Algorithm 3 line 7 guarantees that, after $t_l + T_p$, the system publishes at any message's arrival (no matter it comes from pivot or not). We denote this message as m, and the interval from $t_l + T_p$ to its arrival as T_α. In this way, we guarantees that $t_f^2 - t_f^1$ is bounded by $T_p + T_\alpha$.

Both T_p and T_α are variables that change at each message's arrival, but they can be upper-bounded based on Lemmas 5 and 4, respectively, and the two bounds happens to be equal. This explains the factor 2 in Lemma 6 and Theorem 3. Based on this comprehension, if one can obtain the information of the pivot's mean frequency $\bar{f}_p$ at a time t, a tighter reaction latency upper bound corresponding to a message m_i can be derived

$$\bar{R}_t = A_i + \frac{1}{\bar{f}_p} + \min_{j\in\{1,\ldots,N\}} A_j.$$

The index t of $\bar{R}$ indicates that this bound is a variable depending on the specific value $\bar{f}_p$ at time t.

Moreover, T_p in the above analysis is essentially derived from $\bar{f}_p$, the RHS of the condition we add. We use $\bar{f}_p$ in this condition in order to preserve the aim of the policy: publishing at frequency $\bar{f}_p$. If any other value f is used instead, we can similarly derive a corresponding reaction latency upper bound

$$\bar{R} = A_i + U\left(\frac{1}{f}\right) + \min_{j\in\{1,\ldots,N\}} A_j$$

where $U(1/f)$ denotes the upper bound of $(1/f)$. Since the idea to prove has been presented above, we do not redundantly give the detailed proof here.

VII. Experiments

We conduct experiments to validate our abstract model, confirm our description of timing defect, and evaluate the safety, accuracy and robustness of our upper bounds in Theorems 1–3.

A. Experiment Setting

All our experiments are implemented with ROS 2 system of version *Iron Irwini*. Experiments of model validation and bound property exploration are conducted using wide range of synthetic settings on a desktop computer equipped with an Intel Core i5-11400H CPU running at 2.70GHz. Source codes are available on https://github.com/wuchenhaogit/Latency_Analysis_LatestTimeSynchronizer. We also conduct experiments on a real autonomous vehicle model to validate the timing defect and demonstrate its impact.

In order to easily control the delay of each channel, a modification is made to the source code. We randomly generates the delay of each message within their respective range, and replace the arrival time used by the policy to the sum of each message's timestamp and the generated delay. This modification refrain us from unnecessarily bounding the delay and enables us to implement scenarios of zero delay.

There are 6 different experiment settings:

1) *Random Delay:* Different delay time before messages arrive, ranging from 0 to 40 ms.
2) *Period Ratio:* Different ratio of T^W/T^B, ranging from 1.0 to 8.0. Each T_i^B is randomly selected from a range of 50 ms to 100 ms.
3) *Number of Channels:* Different number of input channels, ranging from 3 to 9.
4) *Frequency Weight:* Different EMA weight of frequency β^f. Each β_i^f is randomly selected from $[0, 1]$.
5) *Error Weight:* Different EMA weight of error β^e. Each β_i^e is randomly selected from $[0, 1]$.
6) *Margin Factor:* Different margin factors γ. Each γ_i is randomly selected from $[0, 64]$.

Each setting is further split into several groups to capture the impact of the corresponding parameter changes.

B. Evaluation Results

We evaluate the model of the *LatestTime* policy presented in Section IV-C. Experiments are conducted under all settings. When messages arrive at the synchronizer, the original policy and our model independently synchronize and publish. We compare their outputs of 2000 published sets. No discrepancy is encountered in 50 repeated experiments. This consistency indicates that our model accurately captures the main features of the *LatestTime* policy with a high level of confidence.

We conduct an experiment on an autonomous vehicle model to confirm the timing defect. The vehicle is equipped with two identical sensors operating at the same frequency. When

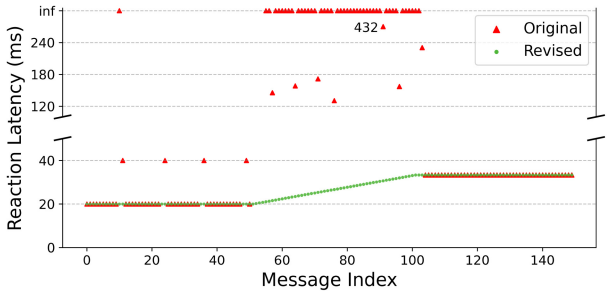

Fig. 7. Reaction latency result of one sensor during frequency decreasing.

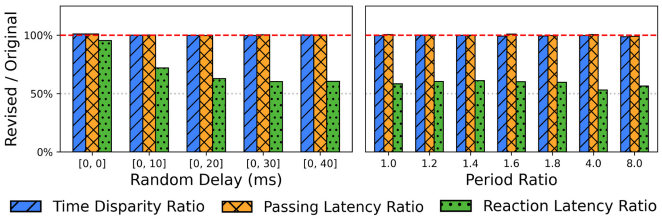

Fig. 8. Result of comparing the revised and original *LatestTime* policy.

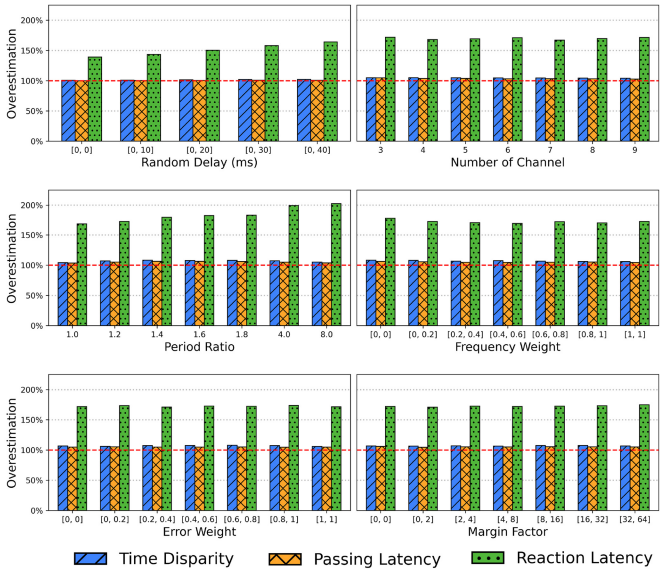

Fig. 9. Evaluation result of the upper bounds of the three metrics.

the car decelerates, frequencies of sensors are lowered based on velocity in order to reduce unnecessary workload. In this scenario the two sensors gradually decrease their frequencies and may trigger the defect. Fig. 7 shows the result of messages from one sensor. Each point represents a message. x-axis is message index in temporal order. y-axis is the reaction latency. Reaction latency of messages that not published is set as infinity. Before index 50, frequencies of both sensor are 50Hz. Due to small fluctuation, pivot occasionally changes between them and might result in long reaction latency of the original synchronizer (red triangles). When frequency gradually decreases to 30Hz, the timing defect is triggered more frequently, leading to severe increase of reaction latency. The worst observed reaction latency is 432 ms, over 20 times of pivot's period. Our revision effectively avoids the defect (green dots).

We conduct experiments to compare the original *LatestTime* policy and the revised policy in Algorithm 3. Results are shown in Fig. 8. The height of bar represents the ratio of the worst observed value of the revised to the original policy. With regard to the time disparity and passing latency, results under all configuration is nearly 100%, which indicates that our revision has no impact on these metrics.

The reaction latency ratio, depicted as the green bar, is 95% and 72% when delay range is $[0, 0]$ and $[0, 10]$ in *Random Delay*. The reason lies in that, the experiments of *Random Delay* are conducted with period ratio set as 1. When delay is very low, the frequency of each channel is rather stable. The pivot seldom changes and the worst case is rarely encountered. In settings of higher delay, the revised policy shows a 40% decrease of the worst observed reaction latency, validating the effectiveness of the revision.

Upper bounds of the three metrics are evaluated based on our revised policy under different experiment settings. In all experiments, there is no underestimation observed, validating the safety of our bounds. Fig. 9 depicts the results. The height of the bar represents the average overestimation of 20 repeated experiments. Overestimation of one experiment is calculated as the ratio of the bound to the worst observed value amongst 2000 published sets.

Both the time disparity and passing latency upper bounds exhibit robustness and low pessimism across various experiment settings, with an average overestimation below 10%. This indicates that the bounds are highly precise and the worst-case scenario is easy to encounter.

The reaction latency upper bound shows an approximately 70% overestimation in different settings. This discrepancy can be attributed to the rarity of encountering the worst-case scenario for reaction latency. In typical systems with stable frequencies, it is uncommon for the pivot to continuously change. So the worst-case scenario is seldom met and contributes little to the average.

VIII. Conclusion

In this article, we model the *LatestTime* synchronization policy in ROS. We uncover its timing defect, which may result in arbitrarily long or even infinite latency in publishing subsequent outputs, and propose a revision to address this problem. Moreover, we formally develop safe and tight upper bounds for *time disparity* and two types of latency incurred due to message synchronization, i.e., the *passing latency* and *reaction latency*. We conduct experiments under different settings to validate our results. Comparing different synchronization policies of ROS will be topic of our future work.

Appendix A
Exponential Moving Average

The EMA [26] is a statistical method used to estimate the local mean value of a variable. It assigns exponentially greater weight to the recent data points.

Definition 4 (EMA): Let θ be a random variable that is sampled at a series of discrete time points, producing the sequence $\{\theta_1, \ldots, \theta_N\}$. $\beta \in [0, 1]$ is a constant weight. Using $EMA_t(\theta)$ to denote the EMA of the first t data points, it can

be recursively calculated by $EMA_t(\theta) = \beta * \theta_t + (1-\beta) * EMA_{t-1}(\theta)$ for $t \in \{2, 3, \ldots, N\}$, with $EMA_1(\theta) = \theta_1$.

APPENDIX B
IMPLEMENTATION OF DATA UPDATE

We first list the definition used in *data update*. In Section IV-A we have defined the frequency corresponding to two consecutive messages in channel i as $f(m_i^x, m_i^{x+1}) = [1/\alpha(m_i^x) - \alpha(m_i^{x+1})]$, the mean value of all the previous f at a time point as $\bar{f}_i$. Now we define the error corresponding to two consecutive messages as $e(m_i^x, m_i^{x+1}) = |f(m_i^x, m_i^{x+1}) - \bar{f}_i|$. The mean error $\bar{e}_i$ is the mean value of all the previous e at a time point. Recall that a user-input constant γ_i is used to estimate the maximal frequency variation $\gamma_i * \bar{e}_i$.

Data update consists of three phases, distinguished by values of $\Theta_i = 1, 2, 3$ for each channel i. Initially $\Theta_i = 1$. S is the newest message set the system maintains. At the beginning, the system simply set the first message of each channel i as $S[i]$. Below we assume $S[i]$ is already set.

Algorithm 4 provides the pseudo code of the *data update*. When a new message m_i arrives, the new frequency f and error e is calculated (lines 11 and 2). Phase 1 indicates that both $\bar{f}_i$ and $\bar{e}_i$ have not been initialized. The system initializes $\bar{f}_i$ as f and change phase to 2 (line 4).

If phase is 2, the channel has valid $\bar{f}_i$ but invalid $\bar{e}_i$. Therefore, the system updates $\bar{f}_i$ using EMA method and initializes $\bar{e}_i$ as e. Its phase is changed to 3 (lines 7 and 8).

If phase is 3, both $\bar{f}_i$ and $\bar{e}_i$ have already been initialized. The system first checks if current statistics are safe. If the new error e falls within the estimated error bound, $e \le \gamma_i * \bar{e}_i$, the statistics are safe and they are updated using EMA method (lines 12 and 13). Otherwise, statistics are deemed unsafe. The system discards all the previous statistics. It restarts the statistical process by treating $S[i]$ as the first message of the channel and m_i the second. Then the system carries out the phase 1 work and change phase to 2 (line 15).

APPENDIX C
CALCULATION DETAIL OF EXAMPLE IN SECTION V-A

In Fig. 4, each message m_1^{x+1} arrives after an interval from m_1^x, whose length is $(1000/[10-(2x+1)\delta])$ms if the *data update* will reboot in phase 3, and $(1000/[10-(2x+11/9)\delta])$ms if the *data update* will not. Similarly, m_2^{x+1} arrives after an interval from m_2^x, whose length is $(1000/[10-(2x+2)\delta])$ms if the *data update* will reboot in phase 3, and $(1000/[10-(2x+20/9)\delta])$ms otherwise.

These lengths ensure that, upon arrival of m_i^x, $\bar{f}_i$ will be $10-(2x-2+i)\delta$Hz, achieving the discussion in Section V-A.

REFERENCES

[1] R. Li, X. Jiang, Z. Dong, J.-M. Wu, C. J. Xue, and N. Guan, "Worst-case latency analysis of message synchronization in ROS," in *Proc. RTSS*, 2023, pp. 185–197.

[2] R. Li, N. Guan, X. Jiang, Z. Guo, Z. Dong, and M. Lv, "Worst-case time disparity analysis of message synchronization in ROS," in *Proc. RTSS*, 2022, pp. 40–52.

[3] "ROS2 iron." Accessed: Sep. 18, 2024. [Online]. Available: https://docs.ros.org/en/rolling/Releases/Release-Iron-Irwini.html#

[4] "The *ExactTime* policy." Accessed: Sep. 18, 2024. [Online]. Available: http://wiki.ros.org/message_filters#ExactTime_Policy

[5] "The *ApproximateTime* policy." Accessed: Sep. 18, 2024. [Online]. Available: http://wiki.ros.org/message_filters#ApproximateTime_Policy

[6] "The *ApproximateEpsilonTime* policy," Accessed: Sep. 18, 2024. [Online]. Available: https://github.com/ros2/message_filters/blob/rolling/include/message_filters/sync_policies/approximate_epsilon_time.h

[7] J. Sun, T. Wang, Y. Li, N. Guan, Z. Guo, and G. Tan, "SEAM: An optimal message synchronizer in ROS with well-bounded time disparity," in *Proc. RTSS*, 2023, pp. 172–184.

[8] H. Rebecq, T. Horstschaefer, and D. Scaramuzza, "Real-time visual-inertial odometry for event cameras using keyframe-based nonlinear optimization," in *Proc. BMVC*, 2017, pp. 1–8.

[9] M. Huber, M. Schlegel, and G. Klinker, "Application of time-delay estimation to mixed reality multisensor tracking," *J. Virtual Real. Broadcast.*, vol. 11, no. 3, pp. 1–22, 2014.

[10] J. Peršić, L. Petrović, I. Marković, and I. Petrović, "Online multi-sensor calibration based on moving object tracking," *Adv. Robot.*, vol. 35, nos. 3–4, pp. 130–140, 2021.

[11] M. R. Nowicki, "Spatiotemporal calibration of camera and 3-D laser scanner," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6451–6458, Oct. 2020.

[12] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," *Sensors*, vol. 21, no. 6, p. 2140, 2021.

[13] Z. Wang, Y. Wu, and Q. Niu, "Multi-sensor fusion in automated driving: A survey," *IEEE Access*, vol. 8, pp. 2847–2868, 2020.

[14] J. Kelly and G. S. Sukhatme, "A general framework for temporal calibration of multiple proprioceptive and exteroceptive sensors," in *Experimental Robotics*. Berlin, Germany: Springer, 2014.

[15] S. Liu et al., "The matter of time—A general and efficient system for precise sensor Synchronization in robotic computing," in *Proc. RTAS*, 2021, pp. 413–416.

[16] 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 *Proc. RTSS*, 2022, pp. 27–39.

[17] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *Proc. EMSOFT*, 2016, pp. 1–10.

[18] C. S. V. Gutiérrez, L. U. S. Juan, I. Z. Ugarte, and V. M. Vilches, "Towards a distributed and real-time framework for robots: Evaluation of ROS 2.0 communications for real-time robotic applications," 2018, *arXiv:1809.02595*.

[19] H. Teper, M. Günzel, N. Ueter, G. von der Brüggen, and J.-J. Chen, "End-to-end timing analysis in ROS2," in *Proc. RTSS*, 2022, pp. 53–65.

[20] A. A. Abdullah, V. Sudharsan, M. W. Kurt, S. Jinghao, and G. Zhishan, "Response time analysis for dynamic priority scheduling in ROS2," in *Proc. DAC*, 2022, pp. 301–306.

[21] C. Randolph, "Improving the predictability of event chains in ROS 2," M.S. thesis, Dept. Embed. Syst., Delft Univ. Technol., Delft, The Netherlands, 2021.

[22] H. Sobhani, H. Choi, and H. Kim, "Timing analysis and priority-driven enhancements of ROS 2 multi-threaded executors," in *Proc. RTAS*, 2023, pp. 106–118.

[23] H. Choi, Y. Xiang, and H. Kim, "PiCAS: New design of priority-driven chain-aware scheduling for ROS2," in *Proc. RTAS*, 2021, pp. 251–263.

[24] 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 *Proc. MOST*, 2023, pp. 71–82.

[25] "The *LatestTime* policy." Accessed: Sep. 18, 2024. [Online]. Available: https://github.com/ros2/message_filters/blob/rolling/include/message_filters/sync_policies/latest_time.h

[26] "Exponential moving average." [Online]. Available: https://en.wikipedia.org/wiki/Exponential_smoothing