Worst-Case Time Disparity Analysis of Message Synchronization in ROS

Ruoxiang Li[1], Nan Guan[1], Xu Jiang[2], Zhishan Guo[3], Zheng Dong[4] and Mingsong Lv[2]

[1]City University of Hong Kong, Hong Kong SAR [2]Northeastern University, China
[3]North Carolina State University, US [4]Wayne State University, US

***Abstract*—Multi-sensor data fusion is essential in autonomous systems to support accurate perception and intelligent decisions. To perform meaningful data fusion, input data from different sensors must be sampled at time points in close propinquity to each other, otherwise the result cannot accurately reflect the status of the physical environment. ROS (Robotic Operating System), a popular software framework for autonomous systems, provides *message synchronization* mechanisms to address the above problem, by buffering messages carrying data from different sensors and grouping those with similar timestamps. Although message synchronization is widely used in applications developed based on ROS, little knowledge is known about its actual behavior and performance, so it is hard to guarantee the quality of data fusion. In this paper, we model the message synchronization policy in ROS and formally analyze its worst-case *time disparity* (maximal difference among the timestamps of the messages grouped into the same output set). We conduct experiments to evaluate the precision of the proposed time disparity upper bound against the maximal observed time disparity in real execution, and compare it with the synchronization policy in Apollo Cyber RT, another popular software framework for autonomous driving systems. Experiment results show that our analysis has good precision and ROS outperforms Apollo Cyber RT in terms of both observed worst-case time disparity and the theoretical bound.**

I. Introduction

Modern autonomous systems, e.g., autonomous vehicles, robots and drones, heavily rely on multi-sensor data fusion to accurately perceive the surrounding physical environment and make intelligent decisions [1], [2]. Usually, the fusion algorithms are developed under the assumption that input data from different sensors are sampled at the same time. However, this assumption rarely holds in reality, due to both the intrinsic hardware characteristics (e.g., different sensors may have different sampling frequencies and there are clock drifts among different sensors) and the software-incurred delay (e.g., due to preprocessing and transfer of the sensor data). In this paper, we use *time disparity* (which will be formally defined in Section II) to describe the temporal inconsistency of input data from different sensor sources. The fusion results will be much less useful or even completely meaningless if the time disparity is too large. For example, in autonomous vehicles, perception of the external environment usually relies on fusion of independent measurements from multiple sensors. If the measurements from two sensors, e.g., a camera and a LiDAR, happened at two substantially different time points, the fusion of their information will not be useful to reconstruct an accurate view of the surrounding environment [3].

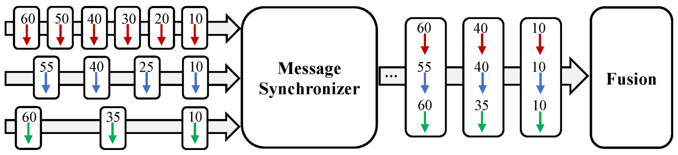


Fig. 1: The role of Message Synchronizer. Arrows denote messages and the number above each arrow denotes its timestamp.

ROS (Robotic Operating System) [4] is a popular software framework for developing robotic systems. ROS has been used by hundreds of thousands of developers to power a large number and different types of robots and other autonomous systems. ROS provides a Message Synchronizer [5] to reduce the time disparity of data inputs to fusion algorithms. As shown in Fig. 1, the Message Synchronizer receives input messages from multiple channels. Each message carries data from some sensor and has a timestamp indicating when the data was sampled. The Message Synchronizer selects one message from each input channel to form an output message set and sends it to the subsequent data fusion component. The Message Synchronizer is widely used in ROS-based applications. For example, Autoware [6], [7], an open-source software framework for autonomous driving systems based on ROS, uses the Message Synchronizer to synchronize camera data and 3D LiDAR data before sending them to the data fusion node. However, although widely used, ROS provides little public information about how its Message Synchronizer actually works, and in particular, what is the exact policy used to select input messages to form the output message set. Therefore, most developers have to use it as a black box, and rely on testing to examine whether the time disparity of its output can meet the fusion algorithm's requirement.

In this paper, we model the ROS Message Synchronizer and formally derive a tight upper bound of the *worst-case time disparity* of its output message sets. Different from testing which only covers a limited part of all possible system behaviors and thus in general is not able to capture the worst-case time disparity, our formal analysis provides an absolute guarantee for the maximal time disparity of the output message set under any circumstance at run time. Therefore, as long as the derived bound falls into the tolerable range of the fusion algorithm, it is ensured that the fusion algorithm will always offer expected quality at run time. Such guarantees are especially important to the design of mission-critical and safe-critical systems, in which (occasional) low-quality outputs may lead to serious consequences.

Our formal analysis is also useful to systems that are less critical and can tolerate occasional low-quality outputs of the fusion algorithm (and thus can tolerate occasional violation of the expected time disparity range). This is because our analysis results provide useful information to efficiently guide system design. In testing, when the observed time disparity is too large, there is no guiding information about how to revise the system design. The common practice is to gradually tune the system parameters and redo the testing, and iterate until the observed time disparity is satisfactory. In contrast, the worst-case time disparity bound derived in this paper has a closed-form relationship with the timing parameters of the sensor data streams, by which we can quickly find the proper range of these parameters to meet the requirement. This is much more efficient than the testing-based approach.

We conduct experiments to evaluate the precision of the time disparity upper bound developed in this paper against the maximal observed time disparity in real execution in ROS with various parameter settings. Experiment results show that our analysis has good precision. We also compare the synchronization policy in ROS with the (simple) policy used in Apollo Cyber RT [8], another popular open-source software framework for autonomous driving systems. Experiment results show that the synchronization policy in ROS outperforms Apollo Cyber RT in terms of both observed worst-case time disparity and the theoretical bound.

II. Problem Model

In this section, we introduce the system model and the problem to be solved in this paper. The system receives inputs from several sensors, each repeatedly generating *messages* carrying sensor data. Each message is associated with a timestamp, which represents the time when its carried sensor data is sampled. Each message may experience some delay (due to, e.g., pre-processing or data transfer) before arriving at the Message Synchronizer. The Message Synchronizer, according to some *synchronization policy*, decides how to select arrived messages from different sensors to form an *output message set* and publishes it. We will introduce the synchronization policy used in ROS Message Synchronizer in Section III. Our target is to analyze the *worst-case time disparity*, i.e., the maximum difference among the timestamps of all messages in an output message set published by the Message Synchronizer. In the following, we will introduce the notations to describe the relevant aspects of the system and then formally define the *worst-case time disparity* metric.

The Message Synchronizer has N input channels. Each channel has a buffer queue Q_i to temporally store the messages arrived at this channel. For this moment, we assume that each queue Q_i is sufficiently long and thus no overflow occurs. Later in Section V, we will get rid of this assumption by providing upper bounds of the required queue size. For simplicity of presentation, we also use Q_i to refer to the

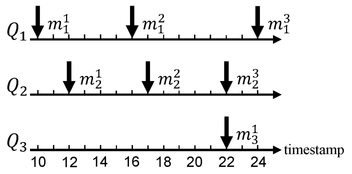


Fig. 2: An example, where the x-axis represents the timestamp.

i^{th} input channel of the Message Synchronizer, when it is unambiguous from the context.

We use m_i^k to represent the k^{th} message currently in queue Q_i. For simplicity, sometimes we also use m_i to represent a message in queue Q_i when there is no need to specify which message it is exactly. Each message m_i^k has a *timestamp*, denoted by $t(m_i^k)$, which represents the time when the sensor data carried by m_i^k was sampled. The difference between the timestamps of two consecutive messages in Q_i is at least T_i^B and at most T_i^W (both T_i^B and T_i^W are strictly greater than 0). For the special case where a sensor periodically generate messages, $T_i^B = T_i^W$. We assume that the messages in the same queue Q_i arrive in the same order as their timestamps, i.e., $\forall k : t(m_i^k) < t(m_i^{k+1})$.

A message may experience some delay before arriving at the Message Synchronizer. Therefore, the timestamp of a message is in general different from the time when the message arrives at the Message Synchronizer. We use D_i^B and D_i^W to represent the best-case and worst-case delay experienced by messages in Q_i, respectively, and use $a(m_i^k)$ to denote the time when message m_i^k arrives at the Message Synchronizer, so we have

$$t(m_i^k) + D_i^B \le a(m_i^k) \le t(m_i^k) + D_i^W$$

The Message Synchronizer selects one message from each queue Q_i to form an *output message set* and publishes it. We say a message set S is a *regular* message set if S has N elements and each element in S comes from a different queue. An output message set published by the Message Synchronizer must be a regular message set. The time disparity of a set of messages is defined as:

Definition 1 (Time Disparity). *Let $S = \{m_1, ..., m_N\}$ be a set of messages. The* time disparity *of S, denoted by $\Delta(S)$, is the maximal difference between the earliest and latest timestamps of messages in S:*

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

For example, in Fig. 2, the time disparity of regular message set $\{m_1^3, m_2^3, m_3^1\}$ is $24 - 22 = 2$, and the time disparity of regular message set $\{m_1^2, m_2^2, m_3^1\}$ is $22 - 16 = 6$.

The problem to solve in this paper is to analyze the worst-case time disparity, i.e., the maximal time disparity of output message sets produced by the Message Synchronizer.

III. ROS Message Synchronization Policy

The ROS Message Synchronizer has two synchronization policies: the *ExactTime* policy [9] and the *ApproximateTime* policy [10]. The *ExactTime* policy is simple: it only selects messages from different input channels with exactly the *same* timestamp to form an output set, and discards messages that cannot find their exact matches. The time disparity of any output message set published under the *ExactTime* policy is trivially 0. However, in reality it is too restrictive to require data from different sensors to have exactly the same timestamp, so the *ExactTime* policy is rarely used in practice. Therefore, we will limit our attention to the widely used *ApproximateTime* policy, which is much more complex and difficult to analyze. Although ROS is evolving actively over years, its Message Synchronizer component is stable. We have checked all ROS 2 C++ versions until the latest *Humble* [11], and ROS 1 C++ versions since *Diamondback* [12], where the message synchronization policies are the same, so the model and results of this paper apply to all these versions.

The *ApproximateTime* policy aims to find and publish regular message sets with as small time disparity as possible. The *ApproximateTime* policy is quite complex and has many optimization and implementation details. It is both infeasible and unnecessary to build a full model equivalent to its source code. Instead, we will develop a high-level abstract model which captures the essential aspects relevant to the analysis problem studied in this paper, and exclude those irrelevant low-level details. In particular, we will only model *what* is its output, rather than *how* the output is obtained in its actual implementation.

It is a natural question whether our abstract model can correctly represent the *ApproximateTime* policy itself. To address this, we conducted intensive experiments to empirically valid our model by comparing the actual outputs of its original implementation in ROS and the expected outputs according to our abstract model. As will be shown in Section VI-A, the actual outputs match the expected outputs according to our model in all experiments. Although this still does not give any absolute assurance, it should be fair to claim the correctness of our abstract model with high confidence. In the following, we present our abstract model, starting with introducing some important concepts used in the *ApproximateTime* policy.

A. Predicted Message

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. A predicted message is never included in output message sets. Instead, it is only used to provide auxiliary information in the selection procedure. m_i^k represents the message stored in the k^{th} position in Q_i, no matter it is an arrived message or a predicted message. If Q_i currently stores messages $\{m_i^1, ..., m_i^k\}$, m_i^k must be a predicted message and $m_i^1, ..., m_i^{k-1}$ are all arrived messages. The timestamp of a *predicted* message m_i^k is set to be

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

where m_i^{k-1} is the last arrived message in Q_i, and T_i^B is the minimal difference between two consecutive messages' timestamps in this channel. When the system starts at time 0, a predicted message with timestamp 0 was initially put into

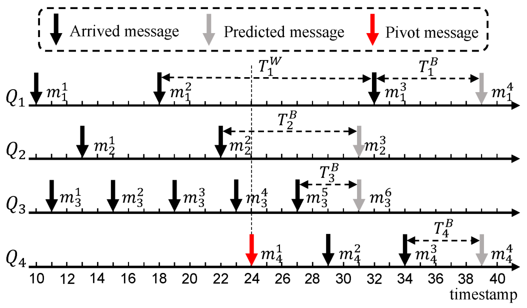


Fig. 3: An example illustrating the *predicted messages*, *pivot* and *selected set*.

each queue. Note that sometime a queue may only have a predicted message but no arrived message.

By considering the predicted messages, the *ApproximateTime* policy can exploit the possibility that the arrived messages may team up with some messages that will arrive in the future to form an output message set with even smaller time disparity. For example, suppose there are two input channels Q_1 and Q_2, with $T_1^B = T_1^W = T_2^B = T_2^W = 10$. Suppose currently there is only one arrived message m_1^1 in Q_1 with $t(m_1^1) = 2$ and only one arrived message m_2^1 in Q_2 with $t(m_2^1) = 10$. If we use m_1^1 and m_2^1 to form an output message set, its time disparity is $t(m_2^1) - t(m_1^1) = 10 - 2 = 8$. However, as messages in Q_1 arrives periodically, the timestamp of the next message to arrive at Q_1 will be $t(m_1^1) + T_1^W = 2 + 10 = 12$, so it is actually better to wait the next message in Q_1 to arrive and let it team up with m_2^1, which results in a smaller time disparity $12 - 10 = 2$.

In general, T_i^W is larger than T_i^B, so the actual timestamp of the next message arrived at Q_1 may be later than the timestamp of the predicted message. For example, we change T_1^W of the above example to 20 and assume that the actual timestamp of the next message of Q_i is 22. In this case, after m_1^2 arrives (and its actual timestamp is known), the *ApproximateTime* policy will select m_1^1 and m_2^1 to form the output message set because teaming up m_1^2 and m_2^1 will lead to larger time disparity, as will be introduced later in this section.

B. Pivot

Definition 2 (Pivot). *Let $S^1 = \{m_1^1, ..., m_N^1\}$, where each m_i^1 is the* first *arrived message 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.

Intuitively, the pivot is a message that must be included in the next published output message set, and other messages will be selected based on how close (in terms of timestamps) they are to the pivot. The pivot changes over time as the status of the queues changes.

The queue containing the pivot is the *pivot queue*, and other queues are *non-pivot queues*. For example, suppose the current queue status is shown in Fig. 3, then m_4^1 is the pivot since it has the latest timestamp among all first messages in all queues. Q_4 is the pivot queue, and Q_1, Q_2, Q_3 are non-pivot queues.

C. Selected Set

The *selected* set is the one with the smallest time disparity among all regular message sets including the pivot.

Definition 3 (Selected Set)**.** *Let Λ be the set of all regular message sets consisting of messages currently in queues (either arrived or predicted) and including the pivot. The* selected set *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 element $S' = \{m'_1, ..., m'_N\}$ in Λ s.t. (i) $\Delta(S') = \Delta(S)$ and (ii) $\exists m_i \in S : t(m'_i) < t(m_i)$.*

If only one regular message set in Λ has the smallest time disparity, it is clearly the selected set. If multiple regular message sets in Λ all have the smallest time disparity, according to the second half of Definition 3, the one with as-early-as-possible messages is the selected set. For example, in Fig. 3, among all regular message sets including the pivot m_4^1, the two sets $S = \{m_1^2, m_2^2, m_3^3, m_4^1\}$ and $S' = \{m_1^2, m_2^2, m_3^4, m_4^1\}$ both have the smallest time disparity 6. According to the second half of Definition 3, S' is *not* the selected set because there exists S containing m_3^3 which is earlier than its correspondence m_3^4 in S'. S is the selected set, because there is no other regular message set containing the pivot has the same time disparity as S and satisfies $\exists m_i \in S : t(m'_i) < t(m_i)$.

Although the selected set is defined as selecting among all regular message sets including the pivot, which could be exponentially many, it is found by a polynomial-time procedure in its implementation in the ROS Message Synchronizer. As we aim to provide a high-level model focusing on *what* is the result generated by the policy, but not *how* the results are obtained, we will not further discuss details of its polynomial-time implementation in ROS.

D. The ApproximateTime *Policy*

Algorithm 1 shows the pseudo-code for the *ApproximateTime* policy to select and publish the output message set when a new message arrives. Suppose at the current moment Q_i originally has k messages $\{m_i^1, ..., m_i^k\}$, where m_i^k is a predicted message (recall that the last message in a queue must be a predicted message). The algorithm first updates Q_i with the newly arrived message m_i (Line 1 to 3), by discarding the original predicted message m_i^k, putting the newly arrived message to the end of Q_i as m_i^k, and finally generating a new predicted message m_i^{k+1} with $t(m_i^{k+1}) = t(m_i^k) + T_i^B$ and put it to the end of Q_i.

Next, the algorithm repeatedly checks whether each queue currently contains at least one arrived message or not. If not, i.e., some queue only has a predicted message, it is impossible to publish an output message set anyway, so the algorithm simply stops without any further checking. If yes, it first sets the current pivot according to Definition 2 (Line 5). Then it

Algorithm 1: Publish output message sets when a new message arrives under the *ApproximateTime* 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
     $t(m_i) + T_i^B$ and put it to the end of $Q_i$ ;
4  while each queue has at least one arrived message do
5      $m_{\mathbf{P}} \leftarrow$ the current pivot (Definition 2);
6      if all predicted messages' timestamps $> t(m_{\mathbf{P}})$
         then
7          $S \leftarrow$ the selected set (Definition 3) ;
8          if all messages in $S$ are arrived messages then
9              publish $S$ ;
10             for each $m_j \in S$ 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;
```

checks whether all predicted messages' timestamps are no earlier than $t(m_{\mathbf{P}})$ (Line 6). If not, the algorithm returns without publishing any output message set (Line 15), the intuition behind which is explained as follows. If a predicted message m_k in some queue Q_k has $t(m_k) < t(m_{\mathbf{P}})$, then the timestamps of the arrived messages (if any) in this queue are even smaller, so this predicted message has the closest timestamp to the pivot $m_{\mathbf{P}}$. Therefore, the next message to arrive in this queue has a chance to make a better output message set than the existing arrived messages in Q_k, so it makes sense to wait until the next message of Q_k arrives (and its actual timestamp is revealed) to make the decision[1].

If all predicted messages' timestamps are no earlier than $t(m_{\mathbf{P}})$, the algorithm will find the selected set S according to Definition 3 (Line 7), and checks whether all messages in S are arrived messages (Line 8). If not, i.e., S contains at least one predicted message, S cannot be published and the algorithm stops (Line 13). If yes, S is published as an output message set (Line 9). After that, for each queue Q_j, the corresponding message m_j in S and all messages in Q_j before m_j are discarded.

From Algorithm 1, we can see that under the *ApproximateTime* policy, the output message sets are decided based on the messages' timestamps, but not their arrival times. The arrival

[1]If the checking of Line 6 is removed, the selected set includes a predicted message and thus cannot be published anyway. Therefore, removing this checking (i.e., removing Line 6, 14 and 15) actually does not change the result of Algorithm 1. However, the checking in Line 6 guarantees the existence of m_i^{Y} when finding the selected set as will be discussed in the Section IV, so we keep it in our abstract model.

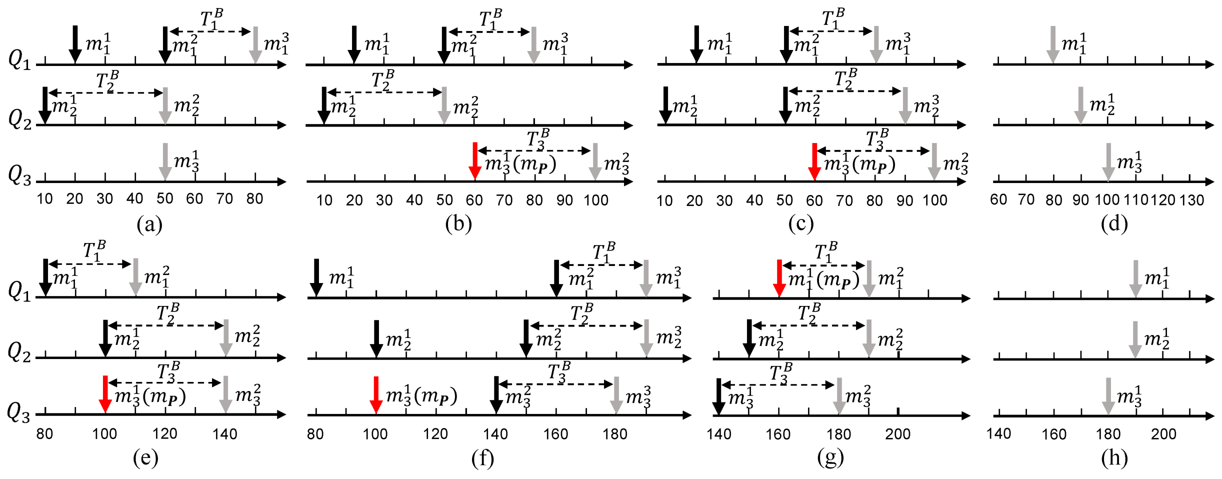


Fig. 4: An example illustrating the *ApproximateTime* policy.

times affect when Algorithm 1 is executed and thus when the output message sets are published.

E. An Illustrative Example

We use Fig. 4 to illustrate Algorithm 1. The x-axis represents the timestamp and the messages' arrival time is not explicitly depicted in the figure. $T_1^B = 30, T_2^B = T_3^B = 40$ and $T_1^W = T_2^W = T_3^W = +\infty$. The original queue status is shown in Fig. 4-(a), and at some later time point a message with timestamp 60 arrives at Q_3. Note that T_3^B is the *minimal* separation between the timestamps of two consecutive messages in Q_3, and this newly arrived message's timestamp is larger than the corresponding predicted message (m_3^1 in Fig. 4-(a)).

This newly arrived message triggers the execution of Algorithm 1. First, Q_3 is updated, discarding the original predicted message, inserting the newly arrived message to Q_3 as the new m_3^1 and generating a new predicted message m_3^2, as shown in Fig. 4-(b). Now each queue has at least one arrived message, so the while-condition in Line 4 is true. m_3^1 is the pivot (denoted by $m_{\mathbf{P}}$) since it has the largest timestamp among the first messages of all queues. Since the timestamp of the predicted message m_2^2 is smaller than $m_{\mathbf{P}}$, the if-condition in Line 6 is false and the algorithm stops.

At some later point, a message with timestamp 50 arrived at Q_2, which triggers the execution of Algorithm 1. After updating Q_2 with this newly arrived message, m_3^1 is still the pivot, as shown in Fig. 4-(c). Now the if-condition in Line 6 is satisfied. The regular message set $S = \{m_1^2, m_2^2, m_3^1\}$ has the smallest time disparity, so it is the selected set. Since this selected set contains only arrived messages (satisfying the if-condition in Line 8), it will be published and the messages in S and those messages earlier than the corresponding message in S are discarded, resulting in Fig. 4-(d).

Algorithm 1 may publish more than one output message set (i.e., iterate for more than one time in the while-loop). Suppose the current queue status is shown in Fig. 4-(e). Now the selected set is $\{m_1^2, m_2^1, m_3^1\}$, which cannot be published since m_1^2 is a predicted message. Later, the next messages in all queues all arrived, but the one in Q_1 has a timestamp much later than predicted, as shown in Fig. 4-(f). Now the selected set is $\{m_1^1, m_2^1, m_3^1\}$, which are all arrived messages and can be published. After that, Algorithm 1 enters the second iteration of the while-loop. Now, as shown in Fig. 4-(g), m_1^1 becomes the pivot and the selected set is $\{m_1^1, m_2^1, m_3^1\}$, which contains only arrived messages and thus can also be published, after which the queue status is shown in Fig. 4-(h).

IV. Time Disparity Analysis

This section derives an upper bound of the time disparity of any output message set published by Algorithm 1. We assume that the timestamps of any two messages (either in the same channel or not) are different. This assumption is only for simplicity of presentation, but does not compromise the generality of our analysis. If two messages indeed have the same timestamp, we can treat them as if one's timestamp is later than the other's by an arbitrarily small amount of time. Our analysis focuses on an arbitrary output message set $S^{\mathbf{PUB}}$ published by Algorithm 1 at some time point. Our target is to upper-bound $\Delta(S^{\mathbf{PUB}})$. We will do this indirectly, by finding a *reference set* S^* with $\Delta(S^*) = \Delta(S^{\mathbf{PUB}})$ and derive an upper bound for $\Delta(S^*)$.

A. Reference Set

Let $m_{\mathbf{P}}$ denote the current pivot. For each non-pivot queue Q_i, we define $m_i^{\mathbf{X}}$ and $m_i^{\mathbf{Y}}$ as:

- $m_i^{\mathbf{X}}$: the ***last*** message in Q_i with $t(m_i) < t(m_{\mathbf{P}})$.
- $m_i^{\mathbf{Y}}$: the ***first*** message in Q_i with $t(m_i) > t(m_{\mathbf{P}})$.

Fig. 5 shows an example illustrating $m_i^{\mathbf{X}}$ and $m_i^{\mathbf{Y}}$. The concepts of $m_i^{\mathbf{X}}$ and $m_i^{\mathbf{Y}}$ only make sense for non-pivot queues. However, we also define $m_i^{\mathbf{X}} = m_i^{\mathbf{Y}} = \mathbf{NULL}$ when Q_i is the pivot queue, where $\mathbf{NULL}$ is an special value that does not equal any message (this enables us to use $m_i^{\mathbf{X}}$ and $m_i^{\mathbf{Y}}$ for all queues, without explicitly distinguishing the difference between pivot and non-pivot queues, which simplifies the presentation).

The definitions of $m_i^{\mathbf{X}}$ and $m_i^{\mathbf{Y}}$ by themselves do not guarantee their existence, i.e., it may be possible that the

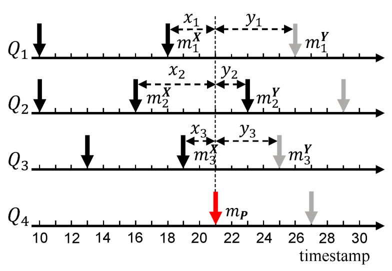


Fig. 5: An example illustrating *neighbour set*, *reference set* and *dominated queue* (Q_3 is dominated by Q_1).

timestamps of messages in Q_i are all smaller (or all larger) than $t(m_\mathbf{P})$. However, when $S^{\mathbf{PUB}}$ is published, $m_i^{\mathbf{X}}$ and $m_i^{\mathbf{Y}}$ indeed both exist:

- If the timestamps of all messages in Q_i are larger than $t(m_\mathbf{P})$, in particular, the timestamp of the first message in Q_i is larger than $t(m_\mathbf{P})$, it contradicts that the pivot's timestamp is the largest among the first messages in all queues. Therefore, $m_i^{\mathbf{X}}$ must exist.
- If the timestamps of all messages in Q_i are smaller than $t(m_\mathbf{P})$, in particular, the timestamp of the last message in Q_i (which is a predicted message) is smaller than $t(m_\mathbf{P})$, it contradicts that Algorithm 1 publishes $S^{\mathbf{PUB}}$ only if all predicted messages' timestamps $> t(m_\mathbf{P})$ (Line 6). Therefore, $m_i^{\mathbf{Y}}$ must exist.

Definition 4 (Neighbor Set)**.** *Let $m_\mathbf{P}$ be the pivot. A* neighbor set *$S = \{m_1, ..., m_N\}$ is a regular message set s.t.,*

1) S includes the pivot $m_\mathbf{P}$ and
2) Each m_i in S, except the pivot, is either $m_i^{\mathbf{X}}$ or $m_i^{\mathbf{Y}}$.

In Fig. 5, $\{m_1^{\mathbf{X}}, m_2^{\mathbf{Y}}, m_3^{\mathbf{X}}, m_\mathbf{P}\}$ and $\{m_1^{\mathbf{Y}}, m_2^{\mathbf{X}}, m_3^{\mathbf{Y}}, m_\mathbf{P}\}$ are two neighbor sets. A neighbor set can choose either $m_i^{\mathbf{X}}$ or $m_i^{\mathbf{Y}}$ for each Q_i ($i = 1, 2, 3$), so there are in total $2^3 = 8$ neighbor sets in this example.

Definition 5 (Reference Set)**.** *A* reference set *S^* has the* smallest *time disparity among all neighbor sets, i.e.,*

$$\Delta(S^*) = \min_{\sigma \in \Omega}\{\Delta(\sigma)\} \tag{1}$$

where Ω is the set of all neighbor sets.

There could be multiple reference sets, which all have the smallest time disparity among the neighbor sets. For example, in Fig. 5, there are two reference sets, $\{m_1^{\mathbf{X}}, m_2^{\mathbf{X}}, m_3^{\mathbf{X}}, m_\mathbf{P}\}$ and $\{m_1^{\mathbf{X}}, m_2^{\mathbf{Y}}, m_3^{\mathbf{X}}, m_\mathbf{P}\}$, both having the smallest time disparity 5.

Lemma 1. *Let $S^{\mathbf{PUB}}$ be a published output message set and S^* the reference set, then*

$$\Delta(S^{\mathbf{PUB}}) = \Delta(S^*) \tag{2}$$

Proof. If $S^{\mathbf{PUB}}$ is a neighbor set, it must be the reference set since it has the smallest time disparity, so the lemma trivially holds. In the following we prove for the case that $S^{\mathbf{PUB}}$ is not a neighbor set.

An output message set must be a selected set, so

$$\Delta(S^{\mathbf{PUB}}) = \min_{S \in \Lambda}\{\Delta(S)\} \tag{3}$$

where Λ is the set of all regular message sets including the pivot. Ω is set of all neighbor sets, and we know $\Omega \subseteq \Lambda$, so by (1) and (3) we know that $\Delta(S^{\mathbf{PUB}}) \leq \Delta(S^*)$.

Then we prove $\Delta(S^*) \leq \Delta(S^{\mathbf{PUB}})$. For an output message set $S^{\mathbf{PUB}} \in \Lambda \setminus \Omega$, we construct a neighbor set $S \in \Omega$: the pivot in $S^{\mathbf{PUB}}$ is also in S; for each non-pivot queue Q_i and $m_i \in S^{\mathbf{PUB}}$:

- if $t(m_i) < t(m_\mathbf{P})$, S includes $m_i^{\mathbf{X}}$.
- if $t(m_i) > t(m_\mathbf{P})$, S includes $m_i^{\mathbf{Y}}$.

For both cases, the selected message from Q_i in S is closer to the pivot in terms of timestamp than the correspondence in $S^{\mathbf{PUB}}$, so $\Delta(S) \leq \Delta(S^{\mathbf{PUB}})$. By (1), we know $\Delta(S^*) \leq \Delta(S) \leq \Delta(S^{\mathbf{PUB}})$. In conclusion, the lemma is proved. □

In the following we will derive an upper bound of $\Delta(S^*)$. We first introduce some auxiliary notations. For each non-pivot queue we define

$$x_i = t(m_\mathbf{P}) - t(m_i^{\mathbf{X}}) \tag{4}$$

$$y_i = t(m_i^{\mathbf{Y}}) - t(m_\mathbf{P}) \tag{5}$$

For a neighbor set $S = \{m_1, \cdots, m_N\}$, we define

$$l(S) = l^{\mathbf{X}}(S) + l^{\mathbf{Y}}(S), \quad \text{where}$$
$$l^{\mathbf{X}}(S) = \max_{m_i = m_i^{\mathbf{X}}}\{x_i\}, \quad l^{\mathbf{Y}}(S) = \max_{m_i = m_i^{\mathbf{Y}}}\{y_i\}$$

For example, in Fig. 5, $S = \{m_1^{\mathbf{Y}}, m_2^{\mathbf{Y}}, m_3^{\mathbf{X}}, m_\mathbf{P}\}$ is a neighbor set, for which we have $l^{\mathbf{X}}(S) = 2$, $l^{\mathbf{Y}}(S) = 5$ and $l(S) = 7$.

Lemma 2. *Let S^* be the reference set and Ω the set of all neighbor sets, then*

$$\Delta(S^*) = \min_{\sigma \in \Omega}\{l(\sigma)\} \tag{6}$$

Proof. For each neighbor set $S = \{m_1, \ldots, m_N\}$:

$$\Delta(S) = \max_{m_j \in S}\{t(m_j)\} - \min_{m_i \in S}\{t(m_i)\}$$
$$= \max_{m_j \in S}\{t(m_j) - t(m_\mathbf{P})\} + \max_{m_i \in S}\{t(m_\mathbf{P}) - t(m_i)\}$$

Since $m_i \neq m_i^{\mathbf{X}} \implies t(m_\mathbf{P}) - t(m_i) < 0$ and $m_i \neq m_j^{\mathbf{Y}} \implies t(m_i) - t(m_\mathbf{P}) < 0$,

$$\Delta(S) = \max_{m_j = m_j^{\mathbf{Y}}}\{t(m_\mathbf{P}) - t(m_j)\} + \max_{m_i = m_i^{\mathbf{X}}}\{t(m_i) - t(m_\mathbf{P})\}$$
$$= \max_{m_j = m_j^{\mathbf{Y}}}\{y_i\} + \max_{m_i = m_i^{\mathbf{X}}}\{x_i\} = l^{\mathbf{Y}}(S) + l^{\mathbf{X}}(S) = l(S)$$

and by (1), the lemma is proved. □

B. Removing the Dominated Queues

With Lemma 2, our remaining task is to derive an upper bound for $\min_{S \in \Omega}\{l(S)\}$. To this end, we first exclude the *dominated queues* from our consideration.

Definition 6 (Dominated Queue)**.** *A non-pivot queue Q_j is a* dominated queue *if there exists a non-pivot queue Q_k s.t.,*

$$x_j < x_k \quad \wedge \quad y_j < y_k$$

In this case, we say Q_j is dominated *by Q_k.*

Definition 7 (Reduced Neighbor Set)**.** *S is a neighbor set. The* reduced neighbor set *corresponding to S, denoted by $\hat{S}$, is obtained by removing all messages in dominated queues from S.*

$S \backslash \hat{S}$ is the subset of messages in S in dominated queues.

For example, in Fig. 5, Q_3 is dominated by Q_1, since $x_3 < x_1$ and $y_3 < y_1$. $S = \{m_1^{\mathbf{Y}}, m_2^{\mathbf{Y}}, m_3^{\mathbf{Y}}, m_{\mathbf{P}}\}$ is a neighbor set, and $\hat{S} = \{m_1^{\mathbf{Y}}, m_2^{\mathbf{Y}}, m_{\mathbf{P}}\}$ is the corresponding reduced neighbor set where $m_3^{\mathbf{Y}}$ in the dominated queue Q_3 is removed.

The definition of $l(\cdot)$, $l^{\mathbf{X}}(\cdot)$ and $l^{\mathbf{Y}}(\cdot)$, which were originally defined for a neighbor set S, can also be applied to $\hat{S}$ or $S \backslash \hat{S}$ in the same way. For example, $l^{\mathbf{X}}(\hat{S})$ equals the maximal x_i among all messages in $\hat{S}$ that selects $m_i^{\mathbf{X}}$; $l^{\mathbf{Y}}(S \backslash \hat{S})$ equals the maximal y_i among all messages in $S \backslash \hat{S}$ that selects $m_i^{\mathbf{Y}}$.

Lemma 3. *Let S^* be a reference set and $\hat{\Omega}$ the set of all reduced neighbor sets, then*

$$\Delta(S^*) = \min_{\sigma \in \hat{\Omega}} \{l(\sigma)\}$$

Proof. Let $\hat{S}$ be a reduced neighbor set with minimal $l(\cdot)$, i.e., $l(\hat{S}) = \min_{\sigma \in \hat{\Omega}} \{l(\sigma)\}$. We construct S based on $\hat{S}$: the pivot in $\hat{S}$ is also in S; for each non-pivot queue Q_i

- if Q_i is *not* a dominated queue, the selection between $m_i^{\mathbf{X}}$ and $m_i^{\mathbf{Y}}$ in S is the same as the selection between $m_i^{\mathbf{X}}$ and $m_i^{\mathbf{Y}}$ in $\hat{S}$.
- if Q_i is a dominated queue, the selection between $m_i^{\mathbf{X}}$ and $m_i^{\mathbf{Y}}$ in S is the same as the selection between $m_j^{\mathbf{X}}$ and $m_j^{\mathbf{Y}}$ in $\hat{S}$, where Q_j is a queue dominating Q_i.

$$\begin{aligned} l(S) =& l^{\mathbf{X}}(S) + l^{\mathbf{Y}}(S) \\ =& \max\left(l^{\mathbf{X}}(\hat{S}), l^{\mathbf{X}}(S \backslash \hat{S})\right) + \max\left(l^{\mathbf{Y}}(\hat{S}), l^{\mathbf{Y}}(S \backslash \hat{S})\right) \end{aligned}$$

By the construction of S, for each message $m_i \in S \backslash \hat{S}$, if $m_i = m_i^{\mathbf{X}}$, then $\exists m_j \in \hat{S}$: $m_j = m_j^{\mathbf{X}}$ and $x_j > x_i$, so we have $\max(l^{\mathbf{X}}(\hat{S}), l^{\mathbf{X}}(S \backslash \hat{S})) = l^{\mathbf{X}}(\hat{S})$. By the same reasoning, we also have $\max(l^{\mathbf{Y}}(\hat{S}), l^{\mathbf{Y}}(S \backslash \hat{S})) = l^{\mathbf{Y}}(\hat{S})$, so we have

$$l(S) = l^{\mathbf{X}}(S) + l^{\mathbf{Y}}(S) = l^{\mathbf{X}}(\hat{S}) + l^{\mathbf{Y}}(\hat{S}) = l(\hat{S}) \tag{7}$$

Next we prove $l(S) = \min_{\sigma \in \Omega} \{l(\sigma)\}$. We prove it by contradiction, assuming $\exists : S' \in \Omega : l(S') < l(S)$. Let $\hat{S}'$ denote the reduced neighbor set corresponding to S'.

$$\begin{aligned} l(S') &= l^{\mathbf{X}}(S') + l^{\mathbf{Y}}(S') \\ &= \max\left(l^{\mathbf{X}}(\hat{S}'), l^{\mathbf{X}}(S' \backslash \hat{S}')\right) + \max\left(l^{\mathbf{Y}}(\hat{S}'), l^{\mathbf{Y}}(S' \backslash \hat{S}')\right) \\ \Longrightarrow & l^{\mathbf{X}}(\hat{S}') + l^{\mathbf{Y}}(\hat{S}') \le l(S') \end{aligned}$$

which together with our assumption $l(S') < l(S)$ and (7) gives

$$l(\hat{S}') = l^{\mathbf{X}}(\hat{S}') + l^{\mathbf{Y}}(\hat{S}') < l(S) = l(\hat{S})$$

which contradicts that $\hat{S}$ has the minimal $l(\cdot)$ among all reduced neighbor sets, so our assumption must be false and thus $l(S) = \min_{\sigma \in \Omega} \{l(\sigma)\}$. Combining this with $l(\hat{S}) = \min_{\sigma \in \hat{\Omega}} \{l(\sigma)\}$, (6) and (7), the lemma is proved. $\square$

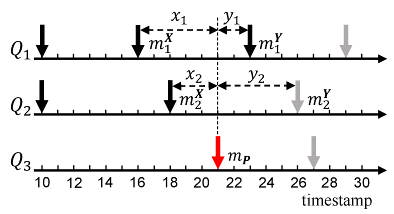


Fig. 6: Illustration for renumbering the queues after removing the dominated queues for the case in Fig. 5.

C. Deriving the Upper Bound

Next we will derive an upper bound for $\min_{\sigma \in \hat{\Omega}} \{l(\sigma)\}$. We use $\hat{N}$ to denote the number of queues in a reduced neighbor set. We renumber[2] the queues so that:

- $Q_1, ..., Q_{\hat{N}-1}$ are non-pivot queues, $Q_{\hat{N}}$ is the pivot queue, and
- $Q_1, ..., Q_{\hat{N}-1}$ are sorted in decreasing order of x_i, i.e.,

$$\forall i \in [1, \hat{N} - 2] : x_i > x_{i+1} \tag{8}$$

Since Q_{i+1} is *not* dominated by Q_i, we also have

$$\forall i \in [1, \hat{N} - 2] : y_i < y_{i+1} \tag{9}$$

For example, after removing dominated queue Q_3 in Fig. 5, the remaining queues are renumbered as shown in Fig. 6.

In the following, we will prove that a reduced neighbor set $\hat{S} = \{\hat{m}_1, ..., \hat{m}_{\hat{N}}\}$ with the minimal $l(\cdot)$ must comply with a particular pattern: if $\hat{m}_i$ selects $m_i^{\mathbf{X}}$, then $\hat{m}_{i+1}, ..., \hat{m}_{\hat{N}-1}$ must also select $m_i^{\mathbf{X}}$, as stated in the following lemma.

Lemma 4. *Let $\hat{S} = \{\hat{m}_1, ..., \hat{m}_{\hat{N}}\}$ be a reduced neighbor set with the minimal $l(\cdot)$ among all reduced neighbor sets. If $\exists \hat{m}_i \in \hat{S} : m_i = m_i^{\mathbf{X}}$, then $\forall k \in [i+1, \hat{N}-1] : \hat{m}_k = m_k^{\mathbf{X}}$.*

Proof. Let i be the smallest index with $\hat{m}_i = m_i^{\mathbf{X}}$ and j largest index with $\hat{m}_j = m_j^{\mathbf{Y}}$ (so $i \neq j$). We will show that $j > i$ leads to a contradiction, so it must be $j < i$ and the lemma holds.

Since i is the smallest index with $\hat{m}_i = m_i^{\mathbf{X}}$ and j is the largest index with $\hat{m}_j = m_j^{\mathbf{Y}}$, by (8) and (9) we know $l^{\mathbf{X}}(\hat{S}) = x_i$ and $l^{\mathbf{Y}}(\hat{S}) = y_j$. If $j > i$, then $x_i > x_j$ and $y_i < y_j$.

Now we consider another $\hat{S}'$, which selects $m_j^{\mathbf{X}}$ instead of $m_j^{\mathbf{Y}}$ and all the other selections are the same as $\hat{S}$. We know that $l^{\mathbf{X}}(\hat{S}') = \max\{x_i, x_j\} = x_i$, i.e., $l^{\mathbf{X}}(\hat{S}') = l^{\mathbf{X}}(\hat{S})$. On the other hand, $l^{\mathbf{Y}}(\hat{S}')$ must become smaller than y_j so $l^{\mathbf{Y}}(\hat{S}') < l^{\mathbf{Y}}(\hat{S})$. In summary, $l(\hat{S}') < l(\hat{S})$, which contradicts that $\hat{S}$ has the smallest $l(\cdot)$ among all reduced neighbor sets in $\hat{\Omega}$. $\square$

The pattern specified in the above lemma can be further divided into three cases, as shown in Fig. 7:

- **Case 1**: $\hat{m}_1, ..., \hat{m}_{\hat{N}-1}$ all select $m_i^{\mathbf{X}}$ (Fig. 7-(a)).
- **Case 2**: $\hat{m}_1, ..., \hat{m}_{\hat{N}-1}$ all select $m_i^{\mathbf{Y}}$ (Fig. 7-(b)).
- **Case 3**: $\exists i$: $\hat{m}_1, ..., \hat{m}_i$ select $m_i^{\mathbf{Y}}$ and $\hat{m}_{i+1}, ..., \hat{m}_{\hat{N}-1}$ select $m_i^{\mathbf{X}}$ (Fig. 7-(c)).

[2]We renumber the queues to simplify the presentation of the following proofs. This does not compromise the generality of our analysis, as one can arbitrarily renumber the queues without affecting the analysis results.

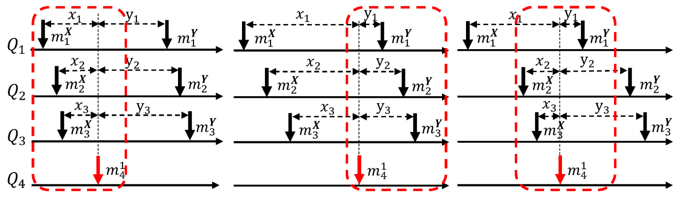

(a) Case 1. (b) Case 2. (c) Case 3.

Fig. 7: The three cases of the pattern specified in Lemma 4.

Now we are ready to upper-bound $\min_{\sigma\in\hat{\Omega}}\{l(\sigma)\}$:

Lemma 5. *Let $\hat{\Omega}$ be the set of all reduced neighbor sets and $\hat{N}$ the number of messages in a reduced neighbor set, then*

$$\min_{\sigma\in\hat{\Omega}}\{l(\sigma)\} \le \frac{1}{\hat{N}}\sum_{i=1}^{\hat{N}-1} T_i^W \tag{10}$$

Proof. Let $\hat{S} = \{\hat{m}_1, ..., \hat{m}_{\hat{N}}\}$ be the one with smallest $l(\cdot)$ among all elements in $\hat{\Omega}$. For each $i \in [1, \hat{N}-2]$, we define

$$\sigma^i = \{m_1^{\mathbf{Y}}, \cdots, m_i^{\mathbf{Y}}, m_{i+1}^{\mathbf{X}}, \cdots, m_{\hat{N}-1}^{\mathbf{X}}, m_{\mathbf{P}}\}$$

where $m_{\mathbf{P}}$ is the pivot. By (8) and (9) we know $\forall i \in [1, \hat{N}-2] : l(\sigma^i) = y_i + x_{i+1}$. We use $\sigma^{\text{all-x}}$ to represent the element in $\hat{\Omega}$ which selects $m_i^{\mathbf{X}}$ for each $i \in [1, \hat{N}-1]$, so $l(\sigma^{\text{all-x}}) = x_1$, and use $\sigma^{\text{all-y}}$ to represent the element in $\hat{\Omega}$ which selects $m_i^{\mathbf{Y}}$ for each $i \in [1, \hat{N}-1]$, so $l(\sigma^{\text{all-y}}) = y_{\hat{N}-1}$.

We will prove that $l(\hat{S})$ is upper-bounded by the RHS of (10) in three cases:

Case 1: $\hat{m}_1, ..., \hat{m}_{\hat{N}-1}$ all select $m_i^{\mathbf{X}}$ (Fig. 7-(a)). All queues are sorted in decreasing order of x_i, so $l(\hat{S}) = x_1$. Since $l(\hat{S})$ is the minimal among all reduced neighbor sets, we know $l(\sigma^{\text{all-y}}) \ge l(\hat{S})$ and $\forall i \in [1, \hat{N}-2] : l(\sigma^i) \ge l(\hat{S})$, i.e.,

$$\begin{cases} x_1 \le y_1 + x_2 \\ \cdots \\ x_1 \le y_{\hat{N}-2} + x_{\hat{N}-1} \\ x_1 \le y_{\hat{N}-1} \end{cases}$$

Adding all these inequalities gives

$$(\hat{N}-1)x_1 \le y_1 + x_2 + y_2 + \cdots + x_{\hat{N}-2} + y_{\hat{N}-2} + x_{\hat{N}-1} + y_{\hat{N}-1}$$

By adding x_1 and dividing $\hat{N}$ on both sides, and combining $x_i + y_i \le T_i^W$ ($\forall i \in [1, \hat{N}-1]$), we get

$$l(\hat{S}) = x_1 \le \frac{1}{\hat{N}}\sum_{i=1}^{\hat{N}-1} T_i^W$$

Case 2: $\hat{m}_1, ..., \hat{m}_{\hat{N}-1}$ all select $m_i^{\mathbf{Y}}$ (Fig. 7-(b)). The proof of this case is symmetric to **Case 1**, so we only briefly sketch it to save space. In this case, $l(\hat{S}) = y_{\hat{N}-1}$, and we know

$$\begin{cases} y_{\hat{N}-1} \le x_1 \\ y_{\hat{N}-1} \le y_1 + x_2 \\ \cdots \\ y_{\hat{N}-1} \le y_{\hat{N}-2} + x_{\hat{N}-1} \end{cases}$$

Putting them together and by $x_i + y_i \le T_i^W$ for each $i \in [1, \hat{N}-1]$, we get

$$l(\hat{S}) = y_{\hat{N}-1} \le \frac{1}{\hat{N}}\sum_{i=1}^{\hat{N}-1} T_i^W$$

Case 3: $\exists i$: $\hat{m}_1, ..., \hat{m}_i$ select $m_i^{\mathbf{Y}}$ and $\hat{m}_{i+1}, ..., \hat{m}_{\hat{N}-1}$ select $m_i^{\mathbf{X}}$ (Fig. 7-(c)). By Lemma 4, we know there is some k such that $\hat{S} = \sigma^k$, so $\forall i \in [1, \hat{N}-2] : l(\sigma^k) \le l(\sigma^i)$. We also have $l(\sigma^k) \le l(\sigma^{\text{all-x}}) = x_1$ and $l(\sigma^k) \le l(\sigma^{\text{all-y}}) = y_{\hat{N}-1'}$. So we have

$$\begin{cases} y_k + x_{k+1} \le x_1 \\ y_k + x_{k+1} \le y_1 + x_2 \\ \cdots \\ y_k + x_{k+1} \le y_{\hat{N}-2} + x_{\hat{N}-1} \\ y_k + x_{k+1} \le y_{\hat{N}-1} \end{cases} \tag{11}$$

by which we get

$$l(\hat{S}) = l(\sigma^k) = x_{k+1} + y_k \le \frac{1}{\hat{N}}\sum_{i=1}^{\hat{N}-1} T_i^W$$

In summary, for all three cases, we have proved

$$l(\hat{S}) \le \frac{1}{\hat{N}}\sum_{i=1}^{\hat{N}-1} T_i^W$$

□

Theorem 1. *The time disparity of a published set $S^{\mathbf{PUB}}$ is upper-bounded by*

$$\Delta(S^{\mathbf{PUB}}) \le \max_{2\le n\le N}\left\{\frac{1}{n}\sum_{n-1 \text{ largest}} T_i^W\right\}. \tag{12}$$

Proof. By Lemma 1, Lemma 3 and Lemma 5, we have

$$\Delta(S^{\mathbf{PUB}}) \le \frac{1}{\hat{N}}\sum_{i=1}^{\hat{N}-1} T_i^W.$$

It is unknown which queues are dominated queues. However, we can assume the $\hat{N}-1$ queues with the largest T_i^W are the non-dominated non-pivot queues to upper-bound $\sum_{i=1}^{\hat{N}-1} T_i^W$, and we know $2 \le \hat{N} \le N$, so the theorem is proved. □

For example, suppose $N = 4$, $T_1^W = 20$, $T_2^W = 30$, $T_3^W = 60$ and $T_4^W = 75$. The time disparity bound in (12) is computed by $\max(\frac{75}{2}, \frac{60+75}{3}, \frac{30+60+75}{4}) = \frac{60+75}{3} = 45$.

We can see that the bound in (12) only depends on T_i^W of each queue, but is unrelated to D_i^B and D_i^W. This is consistent with that the time disparity of an output message set published by Algorithm 1 only depends on the messages' timestamps, but not the delay they experienced.

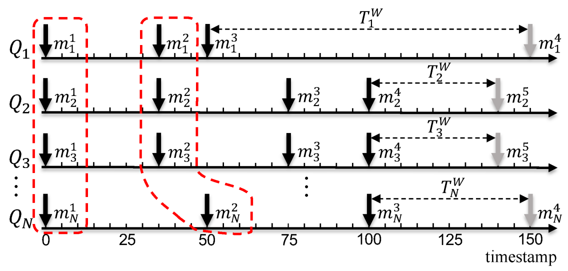


Fig. 8: Illustration for the case of the worst-case time disparity of 50.

D. Tightness

Now we prove that the time disparity bound in Theorem 1 is tight, i.e., for any $N \geq 2$, there exists a system with N input channels whose worst-case time disparity exactly equals the bound in Theorem 1. We can construct the desired system as follows: let $T_1^W = 100$, $T_N^W = 50$ and $T_2^W = ... = T_{N-1}^W = 40$. It is not difficult to see that this system's actual worst-case time disparity is 50. Take the scenario shown in Fig. 8 as an example. After publishing the first two output message sets, the worst-case time disparity of 50 occurs at the third output message set, i.e., $\{m_1^3, m_2^3, m_3^3, ..., m_{N-1}^3, m_N^3\}$. On the other hand, the bound in Theorem 1 equals 50, so its tightness is proved.

Although the above example has already proved the tightness of the bound in Theorem 1, its construction only covers a particular case where the maximal $\frac{1}{n} \sum_{n-1 \text{ largest}} T_i^W$ is achieved with $n = 2$ or $n = 3$. Actually, we can even prove a stronger conclusion in the sense that the bound $\frac{1}{n} \sum_{n-1 \text{ largest}} T_i^W$ is tight for any n, i.e., the bound (10) in Lemma 5 is also tight.

Lemma 6. *The bound (10) in Lemma 5 is tight.*

Proof. Let the system have $N \geq 2$ queues and $\forall i \in [1, N]$: $T_i^W = X$, where X is an arbitrary number that is not too small (e.g., $X > 10$). We construct the worst-case scenario as follows. First, m_N in Q_N is currently the pivot. Second, the timestamps of the first message after the pivot and the last message before the pivot for each non-pivot queue are constructed to satisfy the following constraints:

$$\begin{cases} x_1 = y_1 + x_2 \\ \dots \\ x_1 = y_{N-2} + x_{N-1} \\ x_1 = y_{N-1} \end{cases}$$

i.e., change all the inequalities in Case 1 of the proof of Lemma 5 into equations. By the same reasoning as in the proof of Lemma 5, the one with the minimal time disparity among all reduced neighbor sets must fall in one of the three cases in Lemma 5, i.e., its time disparity is either x_1, or y_{N-1} or $y_{i-1} + x_i$ (for some i), which all equal x_1 by the above equations. Summing up both sides of the above equations gives

$$Nx_1 = x_1 + (y_1 + x_2) + \dots + (y_{N-2} + x_{N-1}) + y_{N-1}$$

On the other hand, we know

$$(x_1 + y_1) + (x_2 + \dots + y_{N-2}) + (x_{N-1} + y_{N-1}) = (N-1)X$$

In summary we have $x_1 = \frac{(N-1)X}{N}$, which exactly equals the bound computed by Lemma 5. $\square$

V. Required Queue Size Bound

The model and analysis introduced so far are based on the assumption that all queues are sufficiently long and no overflow occurs. In this section, we will get rid of this assumption by showing that we can find an upper bound of the queue size with which Theorem 1 still holds.

In reality, the ROS message synchronizer may discard messages for two reasons:

- *Active discard.* Recall that Algorithm 1 (Line 11) discards the published messages and all messages earlier than the published one in each queue. In this case, we say these messages are *actively* discarded.
- *Passive discard.* If a queue is full when a new message arrives, the earliest message in this queue will be discarded. In this case, we say the message is *passively* discarded.

To upper-bound the needed queue size, it seems that we need to find the condition under which passive discard never occurs. However, this is actually unnecessary. Consider two scenarios with the same system input:

S1. Queues sizes are unlimited and no passive discard occurs.
S2. Queues sizes are limited and some messages that are *not* published in **S1** are passively discarded.

The passive discard in **S2** should not affect the selection of pivots and the corresponding selection of published output message sets, and the outputs in these two scenarios are the same. In other words, we can view each unlimited queue in **S1** as having two parts: the first part has the same size as the corresponding queue in scenario **S2**, and the second part has unlimited size and stores the messages that are passively discarded in scenario **S2**. In this case, a message can be published only if it is in the first part of the queue, and messages in the second part will be discarded anyway.

Therefore, in the following, we will calculate the required size of each queue so that when an output message set is published, the queue is large enough to store all messages that contain this published message and ensure the pivot remains unchanged once it is selected, regardless whether the messages before those have been passively discarded or not.

Definition 8 (Required Size of Q_i)**.** $S^{\text{PUB}} = \{m_1, ..., m_N\}$ *is an output message set published at time t. The* required size *of Q_i for S^{PUB} is the number of arrived messages in Q_i, and $m_i \in S^{\text{PUB}}$ satisfies:*

- *if $t(m_i) \leq t(m_{\text{P}})$, m_i is the earliest message in Q_i,* ***or***
- *if $t(m_i) > t(m_{\text{P}})$, there must exist an only message m_i^1 with $t(m_i^1) \leq t(m_{\text{P}})$ that is the earliest message in Q_i.*

We use D_{max}^W and D_{min}^B to denote the maximal D_i^W and minimal D_i^B, and T_{max}^W and T_{min}^B the maximal T_i^W and minimal T_i^B, among all queues Q_i.

Lemma 7. *Let $S^{\mathbf{PUB}} = \{m_1, ..., m_N\}$ be an output message set published at time t. The required size of Q_i for $S^{\mathbf{PUB}}$ is at most $\lfloor \frac{t - a(m_i^1) + D_i^W - D_i^B}{T_i^B} \rfloor + 1$.*

Proof. If a message is in Q_i at time t, its timestamp is no later than $t - D_i^B$. On the other hand, the timestamp of m_i is no earlier than $a(m_i^1) - D_i^W$. Therefore, the total length of the time interval to generate messages that are after m_i^1 and have arrived at Q_i by t is at most $t - a(m_i^1) + D_i^W - D_i^B$. The number of such messages is at most $\lfloor \frac{t - a(m_i^1) + D_i^W - D_i^B}{T_i^B} \rfloor$, and plus m_i^1 itself, the required size of Q_i at t is upper-bounded by $\lfloor \frac{t - a(m_i^1) + D_i^W - D_i^B}{T_i^B} \rfloor + 1$. □

Lemma 8. *Let $S^{\mathbf{PUB}} = \{m_1, ..., m_N\}$ be an output message set published at time t and $m_{\mathbf{P}} \in S^{\mathbf{PUB}}$ is the pivot. Then for each $m_i \in S^{\mathbf{PUB}}$ and m_i^1 is the corresponding earliest message in Q_i, we have*

$$a(m_{\mathbf{P}}) - a(m_i^1) \leq \overline{\Delta} + T_i^W + D_{max}^W - D_i^B$$

where $\overline{\Delta}$ denotes the RHS of (12).

Proof. By Theorem 1, for each $m_j \in S^{\mathbf{PUB}}$, we know $t(m_j) - t(m_i) \leq \overline{\Delta}$. Since $t(m_i) \leq t(m_i^1) + T_i^W$, we have $t(m_j) - t(m_i^1) \leq \overline{\Delta} + T_i^W$. On the other hand, we also have $a(m_j) - t(m_j) \leq D_j^W$ and $a(m_i^1) - t(m_i^1) \geq D_i^B$. Putting them together proves

$$a(m_j) - a(m_i^1) \leq \overline{\Delta} + T_i^W + D_{max}^W - D_i^B$$

Since $m_{\mathbf{P}} \in S^{\mathbf{PUB}}$, the lemma is proved. □

Lemma 9. *Let $S^{\mathbf{PUB}} = \{m_1, ..., m_N\}$ be an output message set published at time t and $m_{\mathbf{P}}$ is the pivot in $S^{\mathbf{PUB}}$, then*

$$t - a(m_{\mathbf{P}}) \leq T_{max}^W + D_{max}^W - D_{min}^B$$

Proof. Let t' denote the earliest time point at which each non-pivot contains at least one *arrived* message with timestamp larger than $t(m_{\mathbf{P}})$. By the definition of t', some message arrives at t' and thus Algorithm 1 is executed at t'. We will first prove $S^{\mathbf{PUB}}$ is published no later than t'. We prove this by contradiction, assuming $S^{\mathbf{PUB}}$ is published after 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, so the selected set at t' must not be $S^{\mathbf{PUB}}$ (otherwise $S^{\mathbf{PUB}}$ is published at t'). Let S be the selected set at t'. First, S does not contain any predicted message, since for each Q_i there exists an arrived message with timestamp later than $t(m_{\mathbf{P}})$ (so the predicted message is "further away" from the $m_{\mathbf{P}}$ than this arrived message). Therefore, the selected set S must be published, so the pivot $m_{\mathbf{P}}$ must not be in S (since the same message cannot be included in two published output message sets). Therefore, for the queue of $m_{\mathbf{P}}$, S includes a message after $m_{\mathbf{P}}$. 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, our assumption is false, so $S^{\mathbf{PUB}}$ is published no later than t', i.e., $t \leq t'$.

We assume message m_i' of Q_i arrived at t' and triggers the execution of Algorithm 1, so m_i' is the first message in Q_i with timestamp $t(m_i') \leq t(m_{\mathbf{P}}) + T_i^W$. Assume the pivot $m_{\mathbf{P}}$ is in queue Q_k, then $t(m_{\mathbf{P}}) \leq a(m_{\mathbf{P}}) - D_k^B$, so in summary we have $t(m_i') \leq a(m_{\mathbf{P}}) - D_k^B + T_i^W$. On the other hand, since $a(m_i') = t'$, we know $t(m_i') \geq t' - D_i^W$. Putting them together, we have $t' \leq a(m_{\mathbf{P}}) + T_i^W + D_i^W - D_k^B \leq a(m_{\mathbf{P}}) + D_{max}^W + T_{max}^W - D_{min}^B$, and since $t \leq t'$, the lemma is proved. □

Theorem 2. *The required size of Q_i for any published output message set is upper bounded by*

$$\left\lfloor \frac{\overline{\Delta} + T_{max}^W + T_i^W + 2D_{max}^W + D_i^W - D_{min}^B - 2D_i^B}{T_i^B} \right\rfloor + 1$$

where $\overline{\Delta}$ denotes the RHS of (12).

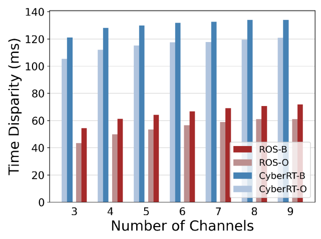
Proof. The theorem is proved by combining Lemma 7, Lemma 9 and Lemma 8. □

VI. Experiments

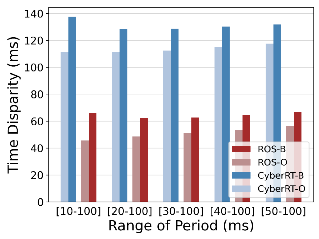
We conduct experiments to both validate our high-level model of the *ApproximateTime* policy and evaluate the analysis precision of the time disparity bound in Theorem 1. The source codes of all experiments are anonymously available online at *https://github.com/ruoxianglee/synchronizer*.

A. Model Validation

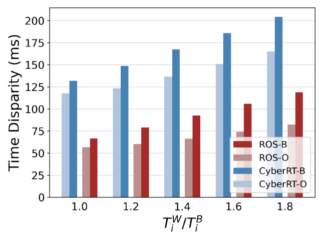
We implement Algorithm 1 (called our implementation) in the Message Filter package of ROS2 (the *Dashing* version) and let it run in parallel with the original implementation of the *ApproximateTime* policy in ROS2. We implement Algorithm 1 in a straightforward way, without any performance optimization, to reduce the chance of introducing implementation errors. When a new message arrives at the Message Synchronizer, our implementation and the original implementation will update their own queues, select and publish the output message set independently. We compare all the output message sets published in the two implementations to see if they are the same. We run the experiments on an Intel i7 desktop computer with ROS2 *Dashing* installed on Ubuntu 18.04, using artificial input messages generated using timers with different settings, including different number of input channels (from 2 to 9, as currently the ROS Message Filter supports up to 9 input channels), different timestamp separation of each channel (T_i^B chosen between 10ms and 100ms, and the ratio between T_i^B and T_i^W chosen between 1 and 1.8). For the experiments in each setting, the delay experienced by the messages randomly varies between 1ms and 40ms. We in total conduct experiments with 700 different settings, and run the system for 0.5 hours in each setting. In all these experiments, the output message sets produced by our implementation and the original implementation are exactly the same. Besides artificial input messages, we also conduct experiments with sensor data inputs generated by the SVL simulator [13] (including camera, LiDAR and IMU sensors in SVL, with different frequency settings), where the outputs of the two implementations are also the same. These experiments justify the correctness of our model with high confidence.



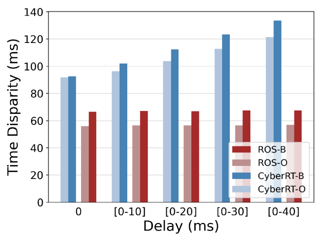
(a)



(b)



(c)



(d)

Fig. 9: Evaluation experiment results.

B. Evaluation of the Time Disparity Bound

We conduct experiments to evaluate the precision of the time disparity bound in Theorem 1 by comparing it with the maximal *observed* time disparity in the real execution in ROS. We also compare with the time disparity (bound) of output produced by the message synchronizer in Apollo CyberRT [8], another popular open source runtime framework for autonomous driving systems. Apollo CyberRT uses a simple synchronization policy: A fixed input channel is selected to be *master channel*, and other channels are *slave channels*. When a new message arrives in the master channel, if all slave channels are not empty, the newly arrived message together with the *latest* message in each slave channel are published as an output message set. No output message set is published upon the arrival of messages in slave channels. For this simple synchronization policy, we can also easily calculate its worst-case time disparity bound (proof provided in the appendix):

$$\max\left\{\max_{2\le i\le N}\{T_i^W+D_i^W\}-D_1^B, D_1^W-\min_{2\le i\le N}\{D_i^B\}\right\} \quad (13)$$

where Q_1 is the master channel. Different from the *ApproximateTime* policy in ROS, the time disparity of output messages under the Apollo CyberRT synchronization policy depends on the delay experienced by the messages.

For each experiment, we compare the following values:

- **ROS-B**: the time disparity bound under the ROS *ApproximateTime* policy calculated by (12) in Theorem 1.
- **ROS-O**: the maximal observed time disparity under the ROS *ApproximateTime* policy in real execution.
- **CyberRT-B**: the time disparity bound under the synchronization policy in Apollo CyberRT calculated by (13).
- **CyberRT-O**: the maximal observed time disparity under Apollo CyberRT synchronization policy in real execution.

Fig. 9-(a) shows the experiment results with different numbers 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 $[1, 40]$. For each x-axis value, the result is the average of 1000 experiments. Each experiment in Fig. 9-(b) uses the same setting as Fig. 9-(a), but sets the number of channels to be 6 and changes the periods as indicated by the x-axis. In Fig. 9-(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. 9-(d), we use the same setting as in Fig. 9-(a), but sets the number of channels to be 6 and changes the range of delay experienced by each message.

From the experiment results we can see that, our time disparity bound in Theorem 1 in general has good precision, but the pessimism increases as the timestamp separation falls in a wider range or the ratio between T_i^W and T_i^B becomes larger. Note that the maximal observed time disparity in real execution only reflects a *lower* bound of the real worst-case time disparity because there is no guarantee to capture the real worst case. Therefore, the actual gap between the real worst-case time disparity and the derived upper bounds could be smaller than the gap indicated in Fig. 9.

As expected, **ROS-B** and **ROS-O** both remain stable as the delay increases, while **CyberRT-B** and **CyberRT-O** both increase as the delay increases. The performance of the ROS *ApproximateTime* policy is significantly better than the simple policy in Apollo CyberRT, in terms of both maximal observed time disparity and the worst-case time disparity bound.

VII. Related Work

Data fusion algorithms are usually developed under the assumption that data from different sensors are perfectly aligned, which rarely holds in reality. To solve this problem, various techniques have been developed to compensate the temporal inconsistency of input data [14]–[17]. However, such compensation works only if the temporal inconsistency falls into a certain range, which is the motivation of this paper.

Previous work [1]–[3], [18] studied how to precisely timestamp the sensor data in the context of multi-sensor data fusion. In this paper, we assume that sensor data are already associated with valid timestamps in the same coordinate using these existing techniques, and focus on the problem after that, i.e., how to manage the sensor data flows in the computing system based on these timestamps.

To promote ROS in time-sensitive application domains, work has been done on measurement-based evaluation of the real-time performance of ROS. [19] evaluated the capabilities and performance for ROS1 and ROS2 with different DDS implementations, considering various metrics, such as latency, throughput, the number of threads and memory consumption. [20] conducted communication evaluation for ROS2 real-time applications taking into account the worst-case latency. To overcome the bottleneck of performance analysis in robot software development, [21] proposes a multipurpose low-overhead framework for tracing ROS application.

Some work aimed to improve the real-time capability of ROS from the system architecture perspective. [22] presented a real-time ROS architecture for separately executing real-time and non-real-time tasks on a integrated OS environment with multi-core processors. [23] proposed an offline scheduling framework for ROS considering both ROS scheduling restrictions and CPU/GPU coordination mechanism. [24] presented a priority-based message transmission mechanism to reduce the worst-case execution time for node processing and inserting a sync node to harmonize the frequencies of different sensor data to improve the time disparity. In [25], a fixed-priority based DAG scheduling framework was proposed with end-to-end latency guarantees. The authors also introduced a synchronization mechanism to reduce the time disparity, but their work is based on measurement for the specific case but does not provide any formal analysis.

Some recent work has been done on formal real-time performance analysis of ROS2. [26], [27] modeled the single-thread Executor in ROS2 and studied response time analysis of processing chains executing on it. [28] redesigns the ROS2 executor with a fixed priority assignment policy to overcome the limitations of the default scheduling strategy of ROS2, and analyze the end-to-end latency based on the proposed scheduling policy. [29] proposes an automatic latency manager and apply existing real-time scheduling theory to latency control of the critical callback chains in ROS2 applications, which adaptively estimates and adjusts the scheduling parameters without the user's involvement. In [30], the authors take both the starvation freedom and execution-time variance of the default ROS2 scheduler into consideration, and propose a more accurate response time analysis for processing chains. [31] presents two new executors based on the thread dispatch model and producer-consumer model and developed corresponding response time analysis techniques. The above work all focus on the executor component in ROS, while in this paper we consider another important component: the Message Synchronizer.

Past work on real-time scheduling and analysis studied different real-time performance metrics, such as response time [32], [33], tardiness [34] and data freshness [35]. However, existing analysis and design techniques developed oriented to these constraints do not apply to the analysis of *time disparity* studied in this paper.

VIII. Conclusion

In this paper, we model the *ApproximateTime* message synchronization policy in ROS and formally analyze the worst-case *time disparity* of their output message sets. We conduct experiments to evaluate the precision of the developed time disparity upper bound against the maximal observed time disparity in real execution, and compare them with the synchronization policy in Apollo CyberRT. Experiment results show that our analysis has good precision and the synchronization policy in ROS greatly outperforms Apollo CyberRT in terms of both observed worst-case time disparity and the theoretical bound. This is the first step towards the analytical study of the data synchronization in multi-sensor data fusion regarding the worst-case time disparity metrics, and many problems along this direction are still open. For example, the required queue size bound derived in this paper is only to show that our time disparity analysis is applicable without assuming infinite queue sizes, but it is unclear whether we could develop tighter bound than that, which will be a topic for our future work. We will also study how to improve the design and implementation of the ROS Message Synchronizer for average-case time disparity performance while maintaining the same (or even better) worst-case time disparity bound.

Acknowledgement

This work is supported by the Research Grants Council of Hong Kong (GRF 11208522, 15206221) and the National Natural Science Foundation of China (NSFC 62102072). The authors also thank the anonymous reviewers for their helpful comments.

Appendix

Theorem 3. *Under the Apollo CyberRT synchronization policy, Q_1 is the master channel, and $Q_2, ..., Q_N$ are slave channels. The time disparity $\Delta(S^{\mathbf{PUB}})$ of a published output message set $S^{\mathbf{PUB}} = \{m_1, ..., m_N\}$ is upper-bounded by:*

$$\Delta(S^{\mathbf{PUB}}) \leq \max\left\{\max_{2\leq i\leq N}\{T_i^W + D_i^W\} - D_1^B, D_1^W - \min_{2\leq i\leq N}\{D_i^B\}\right\}$$

Proof. For m_1 we know

$$a(m_1) - D_1^W \leq t(m_1) \leq a(m_1) - D_1^B \tag{14}$$

For an arbitrary slave queue Q_i ($2 \leq i \leq N$), let m_i' be the next message after m_i. We have

$$t(m_i) + D_i^B \leq a(m_i) \tag{15}$$

$$a(m_1) \leq a(m_i') \leq t(m_i') + D_i^W \tag{16}$$

$$t(m_i') \leq t(m_i) + T_i^W \tag{17}$$

Combing (14)-(17), we have

$$t(m_i) + D_i^B - D_1^W \leq t(m_1) \leq t(m_i) + T_i^W + D_i^B - D_1^B \tag{18}$$

- If $t(m_1) > t(m_i)$, by (18), we have

$$t(m_1) - t(m_i) \leq T_i^W + D_i^W - D_1^B$$

- If $t(m_1) \leq t(m_i)$, by (18), we have

$$t(m_i) - t(m_1) \leq D_1^W - D_i^B$$

The theorem can be proved by combining these two cases. □

References

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

[2] 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.

[3] 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.

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

[5] "Policy-Based Synchronizer." [Online]. Available: http://wiki.ros.org/message_filters#Policy-Based_Synchronizer_.5BROS_1.1.2B-.5D

[6] 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.

[7] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, 2015.

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

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

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

[11] "ROS2 Humble." [Online]. Available: https://docs.ros.org/en/humble/index.html

[12] "ROS Diamondback." [Online]. Available: https://wiki.ros.org/diamondback

[13] "SVL Simulator." [Online]. Available: https://www.svlsimulator.com/

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

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

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

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

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

[19] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ROS2," in *Proceedings of the 13th International Conference on Embedded Software*, 2016, pp. 1–10.

[20] 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," *arXiv preprint arXiv:1809.02595*, 2018.

[21] C. Bédard, I. Lütkebohle, and M. Dagenais, "ros2_tracing: Multipurpose low-overhead framework for real-time tracing of ros 2," *arXiv preprint arXiv:2201.00393*, 2022.

[22] H. Wei, Z. Shao, Z. Huang, R. Chen, Y. Guan, J. Tan, and Z. Shao, "RT-ROS: A real-time ROS architecture on multi-core processors," *Future Generation Computer Systems*, vol. 56, pp. 171–178, 2016.

[23] Y. Suzuki, T. Azumi, S. Kato, and N. Nishio, "Real-Time ROS extension on transparent CPU/GPU coordination mechanism," in *2018 IEEE 21st International Symposium on Real-Time Distributed Computing (ISORC)*. IEEE, 2018, pp. 184–192.

[24] 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.

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

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

[33] 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.

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

[35] 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 *RTAS*, 2021.