Optimizing Logical Execution Time Model for Both Determinism and Low Latency

Sen Wang[1], Dong Li[1], Ashrarul H. Sifat[1], Shao-Yu Huang[2], Xuanliang Deng[1], Changhee Jung[2], Ryan Williams[1], Haibo Zeng[1]
[1]Virginia Tech, [2]Purdue University
Email: {swang666, dongli, ashrar7, xuanliang}@vt.edu, {huan1464, chjung}@purdue.edu, {rywilli1, hbzeng}@vt.edu,

***Abstract*—The Logical Execution Time (LET) programming model has recently received considerable attention, particularly because of its timing and dataflow determinism. In LET, task computation appears always to take the same amount of time (called the task's LET interval), and the task reads (resp. writes) at the beginning (resp. end) of the interval. Compared to other communication mechanisms, such as implicit communication and Dynamic Buffer Protocol (DBP), LET performs worse on many metrics, such as end-to-end latency (including reaction time and data age) and time disparity jitter. Compared with the default LET setting, the flexible LET (fLET) model shrinks the LET interval while still guaranteeing schedulability by introducing the virtual offset to defer the read operation and using the virtual deadline to move up the write operation. Therefore, fLET has the potential to significantly improve the end-to-end timing performance while keeping the benefits of deterministic behavior on timing and dataflow.**

To fully realize the potential of fLET, we consider the problem of optimizing the assignments of its virtual offsets and deadlines. We propose new abstractions to describe the task communication pattern and new optimization algorithms to explore the solution space efficiently. The algorithms leverage the linearizability of communication patterns and utilize symbolic operations to achieve efficient optimization while providing a theoretical guarantee. The framework supports optimizing multiple performance metrics, and guarantees bounded suboptimality when optimizing end-to-end latency. Experimental results show that our optimization algorithms improve upon the default LET and its existing extensions and significantly outperform implicit communication and DBP in terms of various metrics, such as end-to-end latency, time disparity, and its jitter.

I. Introduction

Logical Execution Time (LET) is a programming model suitable for developing real-time control applications [1]. The basic idea behind LET is to abstract away the scheduling and implementation details of software tasks to facilitate model-level, platform-independent testing and verification. This is done by defining a time interval for each task, called LET interval, such that the task computation always appears to take the same amount of time as the LET interval, regardless of how long it actually takes to finish. In addition, the task communication happens at the boundary of the LET interval: the task always reads at the beginning of the interval and writes at its end. In most literatures [1]–[8], the LET interval spans from the task's release to its deadline (default LET), though shorter LET intervals (flexible LET) are also possible.

The excellent properties of LET have drawn strong interest from academia as well as the automotive industry [4]–[7], [9]. In particular, the time determinism and dataflow determinism of LET have made it an appealing solution as a communication mechanism, especially for multicore platforms [10], as these properties simplify the analysis of systems' temporal behavior, stabilize end-to-end latency variance [11], and make the system's temporal behavior composable [3]. Here, time determinism refers to the fact that communication happens only at predefined time instants. Dataflow determinism refers to the property that in a communication link, a consumer job always reads the output value from the same producer job.

However, as a communication mechanism, LET also introduces significant drawbacks regarding end-to-end latency metrics, including reaction time and data age. These metrics are important for real-time systems' safety. For example, it is usually desirable that the real-time computing system reacts to sensor readings or external events as quickly as possible, imposing a system design that minimizes the reaction time [12]. The default LET only makes the output data available at the job deadline, even if the data may be ready much earlier. Hence, compared with other popular communication mechanisms such as implicit communication and Dynamic Buffer Protocol, the default LET is proven to have a longer data age or reaction time [2].

The flexible LET (fLET) model [7], [13]–[15] provides a potential solution by adjusting the LET interval. The difference between fLET and the default LET is that the release time of a task is delayed by a *virtual offset*, and the reading happens deterministically at this deferred release time (Some literatures [13] only delay the reading time). Similarly, a task's deadline and the write operation are brought forward to an earlier *virtual deadline*. We provide more examples of fLET later in the paper (e.g., Fig. 1). The flexibility of fLET provides big potential for performance improvements (as shown in our experiments). However, how to optimize both the virtual offset and virtual deadline while offering a theoretical guarantee is missing in the literature.

Contributions.

- Propose novel optimization algorithms to optimize *both* the LET intervals' start and finish times.
- Perform symbolic operations of a newly introduced concept, *communication pattern*, which compactly models tasks' possible reading/writing relationships.
- Establish suboptimality bounds when minimizing the data age or reaction time. To the best of our knowledge, this

is the first work that provides the theoretical guarantee for fLET while maintaining fast runtime speed.

- Support minimizing various latency metrics for fLET. To the best of our knowledge, this is the first optimization algorithm that supports optimizing time disparity and jitter.
- To the best of our knowledge, this is the first work that shows LET after optimization outperforms alternative communication protocols (implicit communication and DBP) across multiple metrics, such as end-to-end latency, time disparity, and jitter. Our findings enhance fLET's competitiveness in broader scenarios.

Paper Organization. Section II summarizes the related work. Section III introduces the system model and reviews the latency metrics and task communication mechanisms. Section IV describes the flexible LET (fLET) model. Section V defines the communication pattern, a new abstraction for task communication that will be leveraged in our optimization algorithms. Section VI gives a general optimization framework for optimizing any metric, while Section VII proposes new theorems to optimize data age and reaction time efficiently. Section VIII presents generalizations, limitations, and possible solutions. We present the experimental results in Section IX and conclude the paper in Section X.

II. Related Work

The RTSS2021 industry challenge [12] comprehensively summarizes essential latency metrics: data age, reaction time, and time disparity. Among them, data age and reaction time (DART) have been studied extensively for analysis and optimization [16]–[24]. As for time disparity, Li *et al*. [25] developed analysis methods for the Robotic Operating System [25], Jiang *et al*. [26] proposed analysis and buffer size design to reduce the worst-case time disparity. However, most of these studies focus on communication mechanisms other than LET or its variants.

The Logical Execution Time (LET) programming model, introduced by Henzinger *et al*. [1] with the programming language Giotto, has gained attention from the automotive industry [3], [4], [7], [11], [27], [28]. It is also integrated into the automotive software architecture standard AUTOSAR [2], [15]. The popularity in the automotive domain inspires related research in processor assignments [5], [6], end-to-end latency analysis [8], [11], [29], [30], and optimization [11], [31], [32]. However, most research on LET assumes the default LET model (LET intervals equal periods), which performs worse in end-to-end latency metrics than implicit communication and DBP [2] and may suffer from larger latency jitter.

Variants of the flexible LET model and heuristics algorithms have been considered in the literature to reduce the end-to-end latency. Martinez *et al*. [11] proposed to assign an offset to tasks upon their first release while keeping the LET intervals the same as the default LET. Bradatsch *et al*. [32] proposed using the worst-case response time as the length of the LET interval, but the reading time remains the same as the default LET model (i.e., no offset). Maia *et al*. [33] set the LET interval of each task from the smallest relative start time to the largest relative finish time based on schedules. Apart from the fLET model modifications, heuristic optimization algorithms have also been proposed to reduce end-to-end latency [11], [33]. However, these algorithms may suffer from scalability issues or cannot provide an optimality guarantee for fLET.

Offset assignments could reduce the jitter of many metrics, such as response time [34], [35] and end-to-end latency [11]. Bini *et al*. [14] proposed to utilize ring algebra to eliminate the jitter of end-to-end latency in LET models. However, the jitter of the time disparity metric is rarely studied.

Compared to previous LET-related studies, this paper *addresses a more general optimization problem in terms of variables or objective functions. It also introduces novel optimization techniques with improved performance and stronger theoretical guarantees*.

III. System Model and Definitions

This paper uses bold fonts to represent a vector or a set and light characters for scalars. The i^{th} element of a set $\boldsymbol{S}$ is denoted as $\boldsymbol{S}(i)$. We may use sub- and super-scripts to distinguish notations, such as q_j^n.

A. System Model

This paper considers a task set $\boldsymbol{\tau}$ of periodic tasks (Generalization to sporadic tasks will be discussed later). Each task τ_i is described by a tuple (C_i, T_i, D_i^{org}), which denotes the worst-case execution time, period, and relative deadline, respectively. We assume $D_i^{org} \leq T_i$. The least common multiple of all the task periods is hyper-period H. Every T_i time units, τ_i releases a job for execution. The q_i-th released job of the task τ_i is usually denoted as J_{i,q_i}. We allow the job index q_i to be negative numbers. For example, $J_{i,-1}$ is released one period earlier than $J_{i,0}$. To simplify notations, we assume the 0^{th} jobs of all the tasks become ready at time 0. However, our optimization algorithms can optimize LET intervals if there is a known release offset.

We assume some worst-case response time analyses (RTA) are available. The system is schedulable if the RTA is no larger than the relative deadline. Ideally, the worst-case RTA should have no dependency or linear dependency on other tasks' release time and deadline (an example is provided in Section IX). Otherwise, please refer to Section VIII-B.

The input of a task τ_j (or a job J_{j,q_j}) could depend on the output of a task τ_i (or a job J_{i,q_i}), which we denote as $\tau_i \to \tau_j$ (or $J_{i,q_i} \to J_{j,q_j}$). The dependency relationships of all the tasks in $\boldsymbol{\tau}$ can be represented as a directed acyclic graph (DAG) $\boldsymbol{G} = (\boldsymbol{\tau}, \boldsymbol{E})$, where $\boldsymbol{E}$ denotes all the edges in the graph. The DAG is not assumed to be fully connected for generality. A cause-effect chain $\mathcal{C}$ is a list of tasks $\{\tau_0, \tau_1 ... \tau_n\}$ where $\tau_i \to \tau_{i+1}$. These tasks are typically indexed based on their causal relationships: τ_0 is usually the source task that depends on no other tasks in $\mathcal{C}$, τ_n is the sink task that no other tasks in $\mathcal{C}$ depend on. Multiple cause-effect chains may share some tasks in a graph $\boldsymbol{G}$. All the cause-effect chains are denoted as $\boldsymbol{\mathcal{C}}$. A job J_{i,q_i}'s reading (writing) time is denoted as $re_{J_{i,q_i}}$ ($wr_{J_{i,q_i}}$).

B. Data Age and Reaction Time

Data age and reaction time (DART) are among the most commonly used metrics to measure end-to-end latency. Given a cause-effect chain $\mathcal{C} = \tau_0 \rightarrow ... \rightarrow \tau_n$, data age measures the longest time a sensor event influences a computation system. In contrast, reaction time measures the maximum delay between the occurrence of the first sensor event and the last event of the chain that depends on the sensor event. Next, we review the DART analysis following Günzel *et al.* [18]:

Definition III.1 (Job chain [18])**.** *A job chain $\mathcal{C}^J$ of a cause effect chain $\mathcal{C}$ is a sequence of jobs $(J_{0,q_0}, ..., J_{n,q_n})$ where the reading data of job $J_{i+1,q_{i+1}}$ is generated by J_{i,q_i}.*

Definition III.2 (Length of a job chain, Len($\mathcal{C}^J$))**.** *The length of a job chain $\mathcal{C}^J = (J_{0,q_0}, ..., J_{n,q_n})$ is $wr_{J_{n,q_n}} - re_{J_{0,q_0}}$.*

When the reader task and the writer task have different periods, issues of under-sampling may arise, where a writer's data cannot reach any reader due to being overwritten by subsequent writers. Conversely, over-sampling may occur, wherein a writer's data is read by multiple readers simultaneously [17]. The following concepts aid in providing a more precise description of these communication relationships.

Definition III.3 (Last-reading job)**.** *Consider two tasks $\tau_i \rightarrow \tau_j$. For any jobs J_{j,q_j}, it has a unique last-reading job $J_{i,\overleftarrow{q_j}}$ that satisfies the following properties:*

$$wr_{J_{i,\overleftarrow{q_j}}} \leq re_{J_{j,q_j}} < wr_{J_{i,\overleftarrow{q_j}+1}} \tag{1}$$

For example, in Fig. 1b, $J_{1,0}$ is $J_{2,3}$'s last-reading job.

Definition III.4 (Immediate backward job chain, [18])**.** *An immediate backward job chain is a job chain $(J_{0,q_0}, ..., J_{n,q_n})$ where each J_{i,q_i} is the last-reading job of $J_{i+1,q_{i+1}}$.*

Definition III.5 (First-reacting job)**.** *Consider two tasks $\tau_i \rightarrow \tau_j$. For any jobs J_{i,q_i}, it has a unique first-reacting job $J_{j,\overrightarrow{q_i}}$ which satisfies the following properties:*

$$re_{J_{j,\overrightarrow{q_i}-1}} < wr_{J_{i,q_i}} \leq re_{J_{j,\overrightarrow{q_i}}} \tag{2}$$

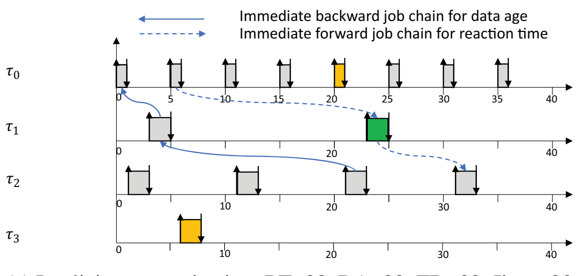
For example, in Fig. 1b, $J_{1,1}$ is $J_{0,0}$'s first-reacting job.

Definition III.6 (Immediate forward job chain, [18])**.** *An immediate forward job chain is a job chain $(J_{0,q_0}, ..., J_{n,q_n})$ where each $J_{i+1,q_{i+1}}$ is the first-reacting job of the job J_{i,q_i}.*

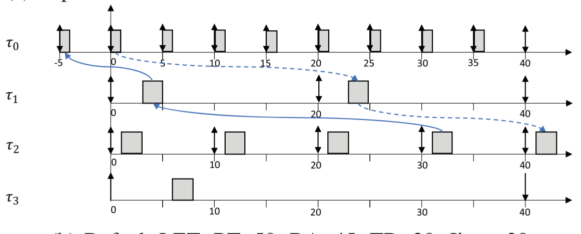
In this paper, We use upper arrows to distinguish one job's last-reading or first-reacting job as in Definition III.3 and Definition III.5. Following [18], the worst-case data age (reaction time) of a cause-effect chain is the length of its longest immediate backward (forward) job chains.

Example 1. Consider a DAG with four tasks: $\{\tau_0, \tau_1, \tau_2, \tau_3\}$ as shown in Fig. 2. The four tasks are scheduled on one CPU with Rate Monotonic preemptively. The objective is minimizing the worst-case data age of the cause-effect chain $\mathcal{C}_0 : \{\tau_0 \rightarrow \tau_1 \rightarrow \tau_2\}$ and $\mathcal{C}_1 : \{\tau_3 \rightarrow \tau_1 \rightarrow \tau_2\}$.

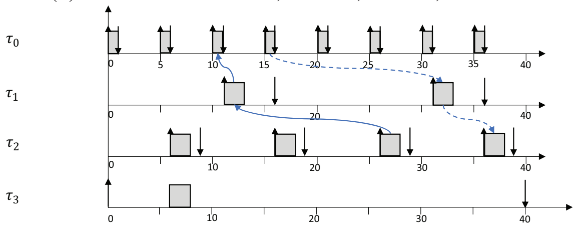
Consider the cause-effect chain $\mathcal{C}_0$. In Fig. 1b, $J_{0,0} \rightarrow J_{1,1} \rightarrow J_{2,4}$ formulate an immediate forward job chain (length is 50, notice $wr_{J_{2,4}} = 50$), $J_{0,-1} \rightarrow J_{1,0} \rightarrow J_{2,3}$ formulate an immediate backward job chain (length is 45).



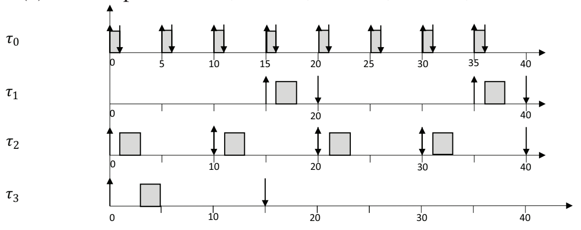
(a) Implicit communication: RT=28, DA=23, TD=33, Jitter=20



(b) Default LET: RT=50, DA=45, TD=20, Jitter=20



(c) fLET-Optimize-DA, RT=**24**, DA=**19**, TD=31, Jitter=20



(d) fLET-Optimize-TD-Jit, RT=35, DA=30, TD=**16**, Jitter=**12**

Figure 1: The schedules of different communication protocols in Example 1. The upward and downward arrows represent job reading and writing times, respectively. Solid leftward arrows connect the longest immediate backward job chains, and dashed rightward arrows connect the longest immediate forward job chains. It is worth highlighting that fLET demonstrates superior performance compared to alternative methods after optimization. Figures 1c and 1d serve merely as illustrations of the optimization results for different metrics. Importantly, our optimization approach supports the simultaneous optimization of multiple metrics.

There are alternative definitions of data age and reaction time [18], [36]. Our optimization algorithm is compatible with them as they have a similar analysis.

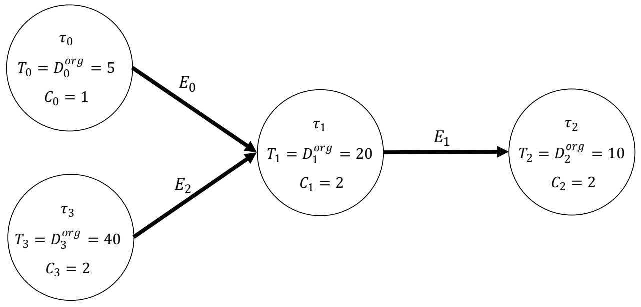


Figure 2: Example DAG with two cause-effect chains.

C. Time disparity

In many situations, such as autonomous driving, a task's input could depend on multiple source tasks (the sink and source tasks constitute a *merge*). In this case, the input data must be collected at approximately the same time to be useful.

Definition III.7 (Time disparity, [12])**.** *Consider a task τ_j which reads input from several source tasks $\boldsymbol{\tau}^s$ (τ_j and $\boldsymbol{\tau}^s$ constitute a "merge"). For each job, J_{j,q_j}, denote its last-reading jobs from $\boldsymbol{\tau}^s$ as $\boldsymbol{J}^{lr,s}$. The time disparity of J_{j,q_j} is defined as the difference between the earliest and the latest writing time of the jobs in $\boldsymbol{J}^{lr,s}$:*

$$TD(J_{j,q_j}) = \max_{J \in \boldsymbol{J}^{lr,s}} wr_J - \min_{J \in \boldsymbol{J}^{lr,s}} wr_J \tag{3}$$

Example 2. In Example 1, τ_1, τ_0 and τ_3 formulate a merge. In Fig. 1a, the job $J_{1,1}$ (highlighted in green) reads data from $J_{0,4}$ (yellow) and $J_{3,0}$ (yellow). The time disparity of $J_{1,1}$ is $21-8 = 13$; Similarly, $J_{1,0}$'s time disparity is $1-(-32) = 33$. τ_1's worst-case time disparity is 33; its jitter is $33 - 13 = 20$.

D. Task Communication Mechanisms

In this section, we briefly review several popular communication mechanisms following [2], [3]:

- Direct Communication: allows unrestricted access to shared variables, enabling tasks to read or write data whenever needed. It may suffer from data inconsistency issues [3].
- Implicit communication: Tasks read input data when they begin execution and write output data when they finish. This approach depends on task schedules, making it more complex to analyze and optimize system behavior, such as end-to-end latency.
- Default Logical Execution Time: The timing of data read and write is deterministic if the system is schedulable, independent of task schedules. Typically, the reading and writing times are defined as follows:

$$\forall J_{i,q_i},\ re_{J_{i,q_i}} = q_i T_i,\ wr_{J_{i,q_i}} = q_i T_i + D_i^{org} \tag{4}$$

- Dynamic Buffer Protocol (DBP): A non-blocking buffering protocol that can preserve synchronous semantics under preemptive scheduling [37]. Instead of restricting data reads and writes at the beginning or end of task execution, DBP allows arbitrary data access through task execution without compromising data consistency by managing multiple buffer copies for the same data. However, there may be additional memory and buffer management overhead.

Under the implicit communication protocol, the length of a cause-effect chain, measured by the maximum data age or reaction time, is no longer than under the DBP communication protocol, which is, in turn, no longer than under the default LET communication protocol [2].

IV. The Flexible LET Model and Problem Description

This section reviews the flexible LET model [7], [13], [15] where tasks can read input and write output at times different from those specified in equation (4):

Definition IV.1 (Virtual offset, O_i)**.** *For each job J_{i,q_i} of a periodic task τ_i, both its release time and the time it reads its input, $re_{J_{i,q_i}}$, is delayed from q_iT_i to $O_i+q_iT_i$, where $O_i \geq 0$.*

Definition IV.2 (Virtual deadline, D_i)**.** *For each job J_{i,q_i} of a periodic task τ_i, both its deadline and the time to write its output, $wr_{J_{i,q_i}}$, is moved forward from $q_iT_i + D_i^{org}$ to $q_iT_i + D_i$, where $O_i \leq D_i \leq D_i^{org}$.*

In the flexible LET model, all jobs of the same task share the same virtual offset and virtual deadline, ensuring deterministic reading and writing times.

A. fLET vs Alternative Communication Mechanism

Compared to alternative methods, such as default LET and implicit communication (both are used in AUTOSAR [2]), fLET shows significant potential in optimizing various performance metrics while still preserving its deterministic advantages. However, realizing these benefits of the flexible LET model depends on finding an optimal set of virtual offsets and virtual deadlines, which is the central focus of this paper.

B. fLET Optimization Problem Descriptions

This paper aims to find virtual offsets and virtual deadlines to minimize the worst-case data age (or reaction time):

$$\min_{\boldsymbol{O},\boldsymbol{D}} \sum_{\mathcal{C}\in\boldsymbol{\mathcal{C}}} \max \boldsymbol{\mathcal{F}}_{\mathcal{C}}(\boldsymbol{O},\boldsymbol{D}) \tag{5}$$

where $\boldsymbol{O}$ and $\boldsymbol{D}$ denote the virtual offset and virtual deadline for all the tasks in the task set $\boldsymbol{\tau}$; $\boldsymbol{\mathcal{F}}_{\mathcal{C}}$ denotes the length of all the immediate forward or backward job chains.

Another important metric considered in this paper is the worst-case time disparity and jitter:

$$\min_{\boldsymbol{O},\boldsymbol{D}} \sum_{\mathcal{M}\in\boldsymbol{\mathcal{M}}} \max \boldsymbol{\mathcal{F}}_{\mathcal{M}}(\boldsymbol{O},\boldsymbol{D}) + \omega(\max \boldsymbol{\mathcal{F}}_{\mathcal{M}}(\boldsymbol{O},\boldsymbol{D}) - \min \boldsymbol{\mathcal{F}}_{\mathcal{M}}(\boldsymbol{O},\boldsymbol{D})) \tag{6}$$

where $\mathcal{M}$ denotes a "merge" that includes a sink task and multiple source tasks that the sink task depends on, $\boldsymbol{\mathcal{F}}_{\mathcal{M}}$ follows Definition III.7, ω is the weight of the jitter term.

Both the optimization problems (5) and (6) are subject to the schedulability constraints:

$$\forall \tau_i \in \boldsymbol{\tau}, 0 \leq O_i,\ O_i + R_i \leq D_i,\ D_i \leq D_i^{org} \tag{7}$$

where R_i denotes the worst-case response time for task τ_i.

V. Communication Patterns

A. Motivation

Directly solving the problem (5) and (6) is challenging due to the nonlinear and non-continuous nature of these problems, which require iterating through all virtual offset and virtual deadline combinations to find the optimal solution. Therefore, this section introduces a new concept, *communication pattern*, which succinctly describes the communication among tasks. We then reformulate the problem (5) and (6) to find the optimal communication pattern. Operations related to the communication patterns are summarized in Table I.

Searching for the communication patterns is easier than value combinations because they have a much smaller solution space. Furthermore, efficient feasibility analysis and symbolic operations are available to reduce unnecessary computation substantially. This method improves both performance and efficiency than the state-of-art [11].

B. Edge Communication Patterns (ECP)

In the flexible LET model, all the jobs of the same task have the same virtual offset and virtual deadline. Therefore, for any edge $E = \tau_i \rightarrow \tau_j$, it is sufficient to focus on jobs within a super-period H^s (the least common multiple of T_i and T_j, [22]) to describe the possible reading/writing relationships.

Definition V.1 (Job map)**.** *A job map is a mapping from one job J_{i,q_i} in a set to another job J_{j,q_j} in another set.*

Definition V.2 (Edge last-reading pattern, ELP)**.** *Consider an edge $E = \tau_i \rightarrow \tau_j$. For any job J_{j,q_j} where $0 \leq q_j < H^s/T_j\}$, the edge last-reading pattern ELP is a job map which specifies which job of τ_i is J_{j,q_j}'s last-reading job.*

Definition V.3 (Edge first-reacting pattern, EFP)**.** *Consider an edge $E = \tau_i \rightarrow \tau_j$. For any job J_{i,q_i} where $0 \leq q_i < H^s/T_i\}$, the edge first-reacting pattern EFP is a job map which specifies which job of τ_j is J_{i,q_i}'s first-reacting job.*

An edge E could have multiple ELPs or EFPs. We use $\boldsymbol{ELP}_E(i)$ ($\boldsymbol{EFP}_E(i)$) to denote the i^{th} ELP (EFP) of E. Both ELP and EFP are considered ECP.

Consider an edge $E = \tau_i \rightarrow \tau_j$, a job J_{j,q_j} and its last-reading job $J_{i,\overleftarrow{q_j}}$, we can use Definition III.3 to get:

$$D_i + \overleftarrow{q_j} T_i \leq O_j + q_j T_j < D_i + (\overleftarrow{q_j} + 1) T_i \tag{8}$$

Therefore, ELPs can be described by linear inequalities:

Definition V.4 (ELP inequalities)**.** *Given an edge $E = \tau_i \rightarrow \tau_j$ and an ELP, the ELP inequalities are the intersection of the linear inequalities (Definition III.3) of all the jobs $J_{j,q_j} \in \boldsymbol{J}_j = \{J_{j,q_j} | 0 \leq q_j < H^s/T_j\}$ and their last-reading jobs:*

$$\mathcal{I}_{ELP} = \mathcal{I}_0^{LR} \cap ... \, \mathcal{I}_{q_j}^{LR} ... \cap \mathcal{I}_{H^s/T_j - 1}^{LR} \tag{9}$$

where $\mathcal{I}_{q_j}^{LR}$ is the linear inequality (8) between J_{j,q_j} and its last-reading job specified by the ELP. The symbol $\cap$ denotes that the two linear inequalities must be *both* satisfied. Notice that Equation (9) can be merged into two linear inequalities since all the $\mathcal{I}_{q_j}^{LR}$ depends on the same two variables (i.e., O_j and D_i). Similarly, EFP inequalities can be obtained based on first-reacting jobs.

Example 3. Consider the edge $E_1 = \tau_1 \rightarrow \tau_2$ in Example 1. Suppose $J_{2,0}$ and $J_{2,1}$'s last-reading jobs are $J_{1,-1}$ and $J_{1,0}$, respectively, then the linear inequalities associated with the two jobs are $D_1 - 20 \leq O_2 < D_1 + 0$ and $D_1 \leq O_2 + 10 < D_1 + 20$. Since these inequalities have to hold together, we can merge them into one set of inequalities by taking the intersection $D_1 - 10 \leq O_2 < D_1$.

Usage. ELPs are used for optimizing data age or time disparity, whereas EFPs are used for optimizing reaction time.

Definition V.5 (Feasibility)**.** *An ELP (EFP) is feasible if there exists a set of virtual offset and virtual deadline assignments that satisfy its inequality constraints (Definition V.4) and schedulability constraints* (7).

Example 4. Consider the edge $E_0 : \tau_0 \rightarrow \tau_1$ in example 1 and an ELP which specifies $J_{1,0}$'s last-reading job to be $J_{0,3}$. It is infeasible because there are no solutions which can satisfy the following inequalities:

$$R_0 \leq D_0, D_0 + 15 \leq O_1 < D_0 + 20, O_1 + R_1 \leq D_1$$

where $R_0 = 1$, $R_1 = 5$ based on RM.

The following lemmas provide the necessary conditions for an edge communication pattern to be feasible.

Lemma 1. *Consider a feasible ELP of an edge $\tau_i \rightarrow \tau_j$ and two jobs J_{j,q_u} and J_{j,q_w} where $q_u < q_w$, their last-reading jobs specified by the ELP, $J_{i,\overleftarrow{q_u}}$ and $J_{i,\overleftarrow{q_w}}$, satisfy: $\overleftarrow{q_u} \leq \overleftarrow{q_w}$.*

Proof. If $\overleftarrow{q_u} > \overleftarrow{q_w}$, then $J_{i,\overleftarrow{q_w}}$ cannot become J_{j,q_w}'s last reading job because J_{j,q_w} could read from $J_{i,\overleftarrow{q_u}}$ since $wr_{J_{i,\overleftarrow{q_w}}} < wr_{J_{i,\overleftarrow{q_u}}} < re_{J_{j,q_u}} < re_{J_{j,q_w}}$. This causes a contradiction. ■

Lemma 2. *Consider a feasible EFP of an edge $\tau_i \rightarrow \tau_j$ and two jobs J_{i,q_u} and J_{i,q_w} where $q_u < q_w$, their first-reacting jobs specified by the EFP, $J_{j,\overrightarrow{q_u}}$ and $J_{j,\overrightarrow{q_w}}$, satisfy $\overrightarrow{q_u} \leq \overrightarrow{q_w}$.*

Proof. The proof is similar to that of Lemma 1. ■

Example 5. Table II shows all the feasible ELPs in example 1 under different situations (since one job could read from different jobs). The mathematical expressions show the intersection of all the possible linear inequalities, following Equation (9). The linear inequalities are merged following Definition V.4.

C. Graph Communication Pattern (GCP)

Given a DAG, we introduce the concept of *Graph Communication Pattern* (GCP) to comprehensively describe the reading and writing relationships among all the tasks in the system.

Definition V.6 (Graph last-reading pattern, GLP)**.** *Consider a DAG $\boldsymbol{G} = (\boldsymbol{\tau}, \boldsymbol{E})$, a graph last-reading pattern $\boldsymbol{GLP}$ is a set of edge last-reading patterns, where for each edge $E_k \in \boldsymbol{E}$, $\boldsymbol{GLP}$ contains one ELP, denoted as $\boldsymbol{GLP}(E_k)$.*

Table I: Edge/Graph last-reading Patterns and operations. (First-reacting patterns are defined similarly)

Symbol	Definition	Mathematical Description	Feasibility Check	Evaluation	Other Operations		
ELP	Job map, Definition V.1	2 Linear Inequalities Definition V.4	Linear inequalities feasibility check Definition V.8	N/A	Add last-reading job pairs, Definition V.4	ELP Comparison, Definition VII.1	N/A
(partial) GLP	ELP set, Definition V.6, Definition V.7	Multiple Linear Inequalities		Definition V.9	Add ELP, Definition V.6	GLP Comparison, Definition VII.2	GLP Contain, Definition VII.3

Table II: Feasible edge last-reading patterns in Example 1

Edge	ELP Pattern Symbol	Last-reading job pairs	Mathematical expression
E_0	$\boldsymbol{ELP}_{E_0}(0)$	$J(0,2) \rightarrow J(1,0)$	$D_0 + 10 \leq O_1 < D_0 + 15$
E_0	$\boldsymbol{ELP}_{E_0}(1)$	$J(0,1) \rightarrow J(1,0)$	$D_0 + 5 \leq O_1 < D_0 + 10$
E_0	$\boldsymbol{ELP}_{E_0}(2)$	$J(0,0) \rightarrow J(1,0)$	$D_0 \leq O_1 < D_0 + 5$
E_0	$\boldsymbol{ELP}_{E_0}(3)$	$J(0,-1) \rightarrow J(1,0)$	$D_0 - 5 \leq O_1 < D_0$
E_1	$\boldsymbol{ELP}_{E_1}(0)$	$J(1,0) \rightarrow J(2,0)$ $J(1,0) \rightarrow J(2,1)$	$D_1 \leq O_2 < D_1 + 3$
E_1	$\boldsymbol{ELP}_{E_1}(1)$	$J(1,-1) \rightarrow J(2,0)$ $J(1,0) \rightarrow J(2,1)$	$D_1 - 10 \leq O_2 < D_1$
E_1	$\boldsymbol{ELP}_{E_1}(2)$	$J(1,-1) \rightarrow J(2,0)$ $J(1,-1) \rightarrow J(2,1)$	$D_1 - 20 \leq O_2 < D_1 - 10$
E_2	$\boldsymbol{ELP}_{E_2}(0)$	$J(3,0) \rightarrow J(1,0)$ $J(3,0) \rightarrow J(1,1)$	$D_3 \leq O_1 < D_3 + 8$
E_2	$\boldsymbol{ELP}_{E_2}(1)$	$J(3,-1) \rightarrow J(1,0)$ $J(3,0) \rightarrow J(1,1)$	$D_3 - 20 \leq O_1 < D_3$
E_2	$\boldsymbol{ELP}_{E_2}(2)$	$J(3,-1) \rightarrow J(1,0)$ $J(3,-1) \rightarrow J(1,1)$	$D_3 - 40 \leq O_1 < D_3 - 20$

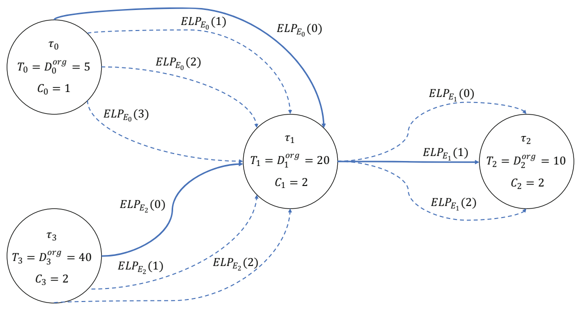


Figure 3: A multi-graph with the edge last-reading patterns (ELPs) in Example 1. A complete graph last-reading pattern (GLP) comprises three ELPs from the three edges. There are $4 \times 3 \times 3 = 36$ possible GLP combinations.

Definition V.7 (Partial GLP)**.** *A partial GLP* $\boldsymbol{pGLP}$ *is a GLP where some or zero edges' ELPs are not included.*

A GLP is complete if the ELPs of all the edges are added; A complete GLP is considered a partial GLP. A GLP is empty if no ELPs are added. The concepts of Graph First-reading Pattern (GFP) and partial GFP are defined similarly. Both GLP and GFP are considered GCP.

Definition V.8 (Feasibility)**.** *A GLP (GFP) is feasible if there is a set of virtual offset and deadline variables that satisfy the linear inequalities of all the ELPs (EFPs).*

Evaluating the feasibility of a GLP (GFP) is equivalent to evaluating the feasibility of a linear programming (LP) problem, which can be efficiently solved to optimality.

D. Evaluating Graph Communication Pattern

Definition V.9 (Graph last-reading pattern evaluation)**.** *A GLP's evaluation, denoted* $\hat{\mathcal{F}}_{\boldsymbol{GLP}}$*, is the optimal objective function values obtained by solving an optimization problem (e.g.,* (5) *or* (6)*) with extra schedulability constraints and linear constraints specified by* $\boldsymbol{GLP}$*.*

For example, if the objective function is given by (5), then:

$$\hat{\mathcal{F}}_{\boldsymbol{GLP}} = \min_{\boldsymbol{O},\boldsymbol{D}} \sum_{\mathcal{C} \in \boldsymbol{\mathcal{C}}} \max \boldsymbol{\mathcal{F}}_{\mathcal{C}}(\boldsymbol{O}, \boldsymbol{D}) \tag{10}$$

$$\forall \tau_i \in \boldsymbol{\tau}, 0 \leq O_i,\ O_i + R_i \leq D_i,\ D_i \leq D_i^{org} \tag{11}$$

$$\forall E_k \in \boldsymbol{E},\ \mathcal{I}_{\boldsymbol{GLP}(E_k)} \tag{12}$$

where $\mathcal{I}_{\boldsymbol{GLP}(E_k)}$ denotes the linear inequality constraint (9) of the edge last-reading pattern $\boldsymbol{GLP}(E_k)$. GFP's evaluation is defined similarly.

Example 6. Continue with Example 1 and we will evaluate $\boldsymbol{GLP} = \{\boldsymbol{ELP}_{E_0}(0), \boldsymbol{ELP}_{E_1}(1), \boldsymbol{ELP}_{E_2}(0)\}$ with only **one** cause-effect chains $\mathcal{C}_0$. Consider a hyper-period of 20 (without considering τ_3), τ_2 has two jobs: $J_{2,0}$ and $J_{2,1}$. All the immediate backward job chains have been decided by $\boldsymbol{GLP}$: $J_{0,-2} \rightarrow J_{1,-1} \rightarrow J_{2,0}$, $J_{0,2} \rightarrow J_{1,0} \rightarrow J_{2,1}$. The objective function becomes:

$$\min_{\boldsymbol{O},\boldsymbol{D}} (\max(wr_{J_{2,0}} - re_{J_{0,-2}},\ wr_{J_{2,1}} - re_{J_{0,2}})) \tag{13}$$

where the reading and writing times are given by Definition IV.1 and IV.2. The objective (13) can be converted into linear functions with extra artificial variables [38]: $a = wr_{J_{2,0}} - re_{J_{0,-2}}$, $b = wr_{J_{2,1}} - re_{J_{0,2}}$, $c = \max(a, b)$:

$$\min_{\boldsymbol{O},\boldsymbol{D}} c \tag{14}$$

$$c \geq a,\ c \geq b \tag{15}$$

The ELP constraints of $\boldsymbol{GLP}$ are adopted from Table II:

$$D_0 + 10 \leq O_1 < D_0 + 15 \tag{16}$$

$$D_1 - 10 \leq O_2 < D_1 \tag{17}$$

$$D_3 \leq O_1 < D_3 + 8 \tag{18}$$

The schedulability constraints (11) remain the same. The optimal solutions of the LP above is $\boldsymbol{O} = \{0, 11, 6, 0\}$, $\boldsymbol{D} = \{1, 16, 9, 40\}$, $\hat{\mathcal{F}}_{\boldsymbol{GLP}} = 19$.

Lemma 3. *Consider a min-max optimization problem*

$$\min_{\boldsymbol{x}} \max(f_1(\boldsymbol{x}), ..., f_n(\boldsymbol{x})) \tag{19}$$

where $f_i(\boldsymbol{x})$ are linear functions. The problem above can be transformed into a linear programming (LP) problem by introducing extra continuous variables for each $f_i(\boldsymbol{x})$.

Lemma 3 shows that evaluating GLPs is usually equivalent to LP. An example of how to perform the transformation is shown in Example 6.

Theorem 1. *Evaluating a GLP when the objective function is data age or time disparity is equivalent to solving an LP.*

Proof. Since the schedulability constraints (11) and all the ELP constraints (Definition V.4) are linear, we will focus on the objective function in the proof. Given a GLP, all the immediate backward job chains are uniquely decided. Therefore, the data age of each immediate backward job chain is a linear function of **O** and **D** (Definition III.2).

As for time disparity, equation (3) can be transformed into linear functions by introducing two continuous artificial variables: $a = \max_{J \in \boldsymbol{J}^{lr,s}} wr_J$, $b = \min_{J \in \boldsymbol{J}^{lr,s}} wr_J$ with extra linear constraints:

$$\forall J \in \boldsymbol{J}^{lr,s}, a \geq wr_J, \ b \leq wr_J \tag{20}$$

The theorem is proved after combining with Lemma 3. ■

Theorem 2. *Evaluating a GFP when the objective function is reaction time is equivalent to solving an LP.*

Proof. Skipped because it is similar to Theorem 1. ■

Observation 1. Evaluating a GLP when the objective function is a time disparity jitter could be equivalent to mixed integer linear programming by following the methods explained in Chapter 3 in Chen *et al*. [38]. How to do it exactly and perform efficient GLP optimization are left as future work.

VI. Two-stage Optimization

Finding the optimal virtual offset and virtual deadline for problem (5) or (6) can be solved in two stages:

- Enumerate all the possible GLPs (GFPs);
- Evaluate (Definition V.9) and select the GLP (GFP) with the minimum objective function value;

The optimal virtual offset and virtual deadline are obtained when evaluating the optimal GLPs (GFPs).

The two-stage method is beneficial because numerous infeasible and non-optimal GLPs can be quickly skipped, as will be introduced next. Since all the definitions and operations of GLP can be symmetrically applied to GFP, we will primarily use GLP to illustrate the algorithms and mention only the necessary changes when optimizing GFPs.

A. Finding Optimal GCP, Enumeration Method

A simple way to find the optimal GLP is to enumerate all the possible ELP combinations. Since each edge E_k has multiple ELPs to select, the enumeration process can be modeled as a multi-graph [39].

Example 7. In Fig. 3, there are 36 GLPs to enumerate.

B. Finding Optimal GCP, Backtracking

Since evaluating GLP's feasibility is equivalent to evaluating linear programming (LP)'s feasibility (Definition V.8), the enumeration method can be expedited using a backtracking algorithm. During enumeration, feasibility is assessed each time a new ELP is added to a partial GLP $\mathbf{P}_{ite}$. If $\mathbf{P}_{ite}$ is infeasible, then no other ELPs will be added to it.

Compared with the enumeration method, the backtracking method avoids evaluating infeasible GLPs while maintaining optimality, resulting in significantly faster speed.

Example 8. Continue with Example 7, 10 out of the 36 possible GLPs are not feasible and will be skipped, such as $\{\boldsymbol{ELP}_{E_0}(0), \boldsymbol{ELP}_{E_1}(1), \boldsymbol{ELP}_{E_2}(0)\}$.

Theorem 3. *The enumeration and the backtracking method above guarantee finding the optimal solutions if not time-out.*

Proof. Since both methods evaluate all the feasible GLPs (GFPs), the solutions are guaranteed to be optimal. ■

VII. DART Symbolic Optimization

This section proposes an efficient backtracking algorithm based on symbolic operations when minimizing data age or reaction time (DART). During iterations, many feasible, but *possibly non-optimal* GLPs (GFPs) will be skipped based on new theorems. The algorithm guarantees that the performance of the final result falls within a small bound of the optimal solution (i.e., bounded suboptimality). The algorithm is motivated by an example shown in Fig. 4.

A. Comparing Edge Communication Patterns

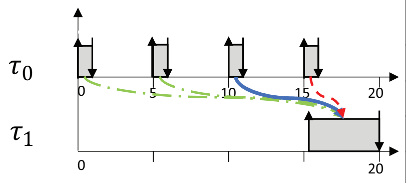
Definition VII.1 (ELP Comparison, smaller, $\preccurlyeq$)**.** *Consider two edge last-reading patterns $\boldsymbol{ELP}_{E_k}(u)$ and $\boldsymbol{ELP}_{E_k}(w)$ for an edge $E_k = \tau_i \to \tau_j$. For any job J_{j,q_j}, if the job index of its last-reading job in $\boldsymbol{ELP}_{E_k}(u)$ is always less than or equal to those in $\boldsymbol{ELP}_{E_k}(w)$, then $\boldsymbol{ELP}_{E_k}(u) \preccurlyeq \boldsymbol{ELP}_{E_k}(w)$.*

Definition VII.2 (GLP Comparison, smaller, $\preccurlyeq$)**.** *If, for each edge E_k, the ELPs of two GLPs satisfy $\boldsymbol{GLP}_u(E_k) \preccurlyeq \boldsymbol{GLP}_w(E_k)$, then $\boldsymbol{GLP}_u \preccurlyeq \boldsymbol{GLP}_w$.*

Similarly, we can define the comparison for EFP and GFP, except we compare the job index of first-reacting jobs.

Example 9. Consider the edge last-reading patterns in Table II. We have $\boldsymbol{ELP}_{E_1}(2) \preccurlyeq \boldsymbol{ELP}_{E_1}(1) \preccurlyeq \boldsymbol{ELP}_{E_1}(0)$. However, consider two imaginary ELPs:

- $\boldsymbol{ELP}_{E_1}(3) : \{J(1,0) \to J(2,0), J(1,2) \to J(2,1)\}$
- $\boldsymbol{ELP}_{E_1}(4) : \{J(1,1) \to J(2,0), J(1,1) \to J(2,1)\}$



Pattern Index	Last-reading job
0	$J_{0,0} \rightarrow J_{1,0}$
1	$J_{0,1} \rightarrow J_{1,0}$
2	$J_{0,2} \rightarrow J_{1,0}$
3	$J_{0,3} \rightarrow J_{1,0}$

Figure 4: Communication pattern comparison. Consider a simple DAG with only one cause-effect chain $\tau_0 \rightarrow \tau_1$. Furthermore, to illustrate the idea more easily, we assume that τ_1's virtual deadline is *locked* at 20. In this case, the optimal edge last-reading pattern (ELP) is $\boldsymbol{ELP}_{E_0}(2)$: $\boldsymbol{ELP}_{E_0}(3)$ is infeasible; $\boldsymbol{ELP}_{E_0}(2)$ has shorter end-to-end latency than $\boldsymbol{ELP}_{E_0}(1)$ and $\boldsymbol{ELP}_{E_0}(0)$ ($\boldsymbol{ELP}_{E_0}(0)$ and $\boldsymbol{ELP}_{E_0}(1)$ require moving τ_1's virtual offset forward). Notice that we can find the optimal ELP based on the pattern index with a feasibility check (a larger index implies a shorter immediate backward job chain).

then neither $\boldsymbol{ELP}_{E_1}(3) \preccurlyeq \boldsymbol{ELP}_{E_1}(4)$ nor $\boldsymbol{ELP}_{E_1}(4) \preccurlyeq \boldsymbol{ELP}_{E_1}(3)$ holds. (Note $\boldsymbol{ELP}_{E_1}(3)$ and $\boldsymbol{ELP}_{E_1}(4)$ are only used for illustration and do not exist in Example 1).

B. Identifying Non-optimal GLPs (GFPs)

Theorem 4. *Consider a cause effect chain $\mathcal{C}$: $\tau_0 \rightarrow \ldots \rightarrow \tau_n$ and two feasible graph last-reading patterns $\boldsymbol{GLP}_u \preccurlyeq \boldsymbol{GLP}_w$. For any job J_{n,q_n}, denote τ_0's job in J_{n,q_n}'s immediate backward job chain in $\boldsymbol{GLP}_u$ and $\boldsymbol{GLP}_w$ as J_{0,q_0^u} and J_{0,q_0^w}, then $q_0^u \leq q_0^w$.*

Proof. We prove the theorem by induction. Following Definition VII.2, the theorem holds when $\mathcal{C}$ has only two tasks. Next, we assume the theorem holds for the cause-effect chain $\tau_i \rightarrow \ldots \rightarrow \tau_n$, and prove that it holds for $\tau_{i-1} \rightarrow \ldots \rightarrow \tau_n$.

Let's denote a job J_{n,q_n}'s last reading job at task τ_i under $\boldsymbol{GLP}_u$ and $\boldsymbol{GLP}_w$ as J_{i,q_i^u} and J_{i,q_i^w}. We know $q_i^u \leq q_i^w$ based on the induction assumption. Next, at task τ_{i-1}, let's denote the last-reading job of J_{i,q_i^u} under $\boldsymbol{GLP}_u$ as $J_{i-1,\overleftarrow{q}_i^u}$, the last-reading job of J_{i,q_i^w} under $\boldsymbol{GLP}_u$ as $J_{i-1,\overleftarrow{q}_i^{wu}}$, the last-reading job of J_{i,q_i^w} under $\boldsymbol{GLP}_w$ as $J_{i-1,\overleftarrow{q}_i^w}$. Then we have $\overleftarrow{q}_i^u \leq \overleftarrow{q}_i^{wu}$ from Lemma 1, and $\overleftarrow{q}_i^{wu} \leq \overleftarrow{q}_i^w$ from the assumption that $\boldsymbol{GLP}_u \preccurlyeq \boldsymbol{GLP}_w$ and Definition VII.1. Therefore, $\overleftarrow{q}_i^u \leq \overleftarrow{q}_i^w$, which proves the induction. ■

Theorem 5. *Consider a cause-effect chain $\mathcal{C}$: $\tau_0 \rightarrow \ldots \rightarrow \tau_n$, and two feasible graph first-reacting patterns $\boldsymbol{GFP}_u \preccurlyeq \boldsymbol{GFP}_w$. Now consider a job J_{0,q_0} of τ_0, denote τ_n's job in J_{0,q_0}'s immediate forward job chain in $\boldsymbol{GFP}_u$ and $\boldsymbol{GFP}_w$ as J_{n,q_n^u} and J_{n,q_n^w}, then $q_n^u \leq q_n^w$.*

Proof. Skipped because it is similar to Theorem 4. ■

Lemma 4. *Continue with Theorem 4, denote the immediate backward job chain that terminates at J_{n,q_n} in $\boldsymbol{GLP}_u$ and $\boldsymbol{GLP}_w$ as $\mathcal{C}_u^J$ and $\mathcal{C}_w^J$. Then we have Len$(\mathcal{C}_w^J)$ – Len$(\mathcal{C}_u^J) \leq B_{\mathcal{C}}$, where $B_{\mathcal{C}} \leq T_n + T_0$.*

Proof. Let's first introduce the related notations:

$$\text{Len}(\mathcal{C}_w^J) = wr^w_{J_{n,q_n}} - re^w_{J_{0,q_0^w}} \tag{21}$$

$$\text{Len}(\mathcal{C}_u^J) = wr^u_{J_{n,q_n}} - re^u_{J_{0,q_0^u}} \tag{22}$$

$$\text{Len}(\mathcal{C}_w^J) - \text{Len}(\mathcal{C}_u^J) = (wr^w_{J_{n,q_n}} - wr^u_{J_{n,q_n}}) + \tag{23}$$

$$- (re^w_{J_{0,q_0^w}} - re^u_{J_{0,q_0^u}}) \tag{24}$$

Since $\boldsymbol{GLP}_u$ and $\boldsymbol{GLP}_w$ are feasible and $q_0^u \leq q_0^w$, we know $re^u_{J_{0,q_0^u}} < re^w_{J_{0,q_0^w}}$ if $q_0^u < q_0^w$, and $|re^w_{J_{0,q_0^w}} - re^u_{J_{0,q_0^u}}| \leq T_0$ if $q_0^u = q_0^w$. Similarly, $wr^w_{J_{n,q_n}} - wr^u_{J_{n,q_n}} \leq T_n$. Therefore, Len$(\mathcal{C}_w^J)$ – Len$(\mathcal{C}_u^J) \leq T_n + T_0$. ■

Theorem 6. *Continue with Theorem 4, the worst-case data age of $\boldsymbol{GLP}_u$ cannot be smaller than $\boldsymbol{GLP}_w$ by more than $B_{\mathcal{C}}$.*

Proof. Let's use two vectors $\mathbf{U}$ and $\mathbf{W}$ to denote the length of the immediate backward job chains in $\boldsymbol{GLP}_u$ and $\boldsymbol{GLP}_w$, respectively. For any job J_{n,q_n}, let's use $\mathbf{U}_{q_n}$ and $\mathbf{W}_{q_n}$ to denote the length of the job chains that terminate at J_{n,q_n}. Following Lemma 4, we have $\mathbf{W}_{q_n} - \mathbf{U}_{q_n} \leq B_{\mathcal{C}}$. Next, let's denote the index of the longest job chain in $\mathbf{U}$ and $\mathbf{W}$ as k_u and k_w. Then we have $\Delta = \mathbf{U}_{k_u} - \mathbf{U}_{k_w} \geq 0$. Therefore,

$$\mathbf{W}_{k_w} - \mathbf{U}_{k_u} = \mathbf{W}_{k_w} - (\mathbf{U}_{k_w} + \Delta) \leq B_{\mathcal{C}} - \Delta \leq B_{\mathcal{C}} \tag{25}$$

■

Theorem 7. *Consider a DAG with multiple cause-effect chains, then Theorem 4 and 6 hold for each cause-effect chain.*

Proof. Because the derivation of the performance bound of each cause-effect chain does not interfere with each other. ■

Example 10. In Example 1, consider two graph last-reading patterns $\boldsymbol{GLP}_u = \{\boldsymbol{ELP}_{E_0}(0), \boldsymbol{ELP}_{E_1}(1), \boldsymbol{ELP}_{E_2}(1)\}$ and $\boldsymbol{GLP}_w = \{\boldsymbol{ELP}_{E_0}(0), \boldsymbol{ELP}_{E_1}(1), \boldsymbol{ELP}_{E_2}(0)\}$. We have $\boldsymbol{GLP}_u \preccurlyeq \boldsymbol{GLP}_w$ following Definition VII.2. Hence, based on Theorem 6, $\hat{\mathcal{F}}_{\boldsymbol{GLP}_w}$ (i.e., the data age after optimization) is guaranteed to be smaller or no larger than $B_{\mathcal{C}}$ (Lemma 4) when compared to $\hat{\mathcal{F}}_{\boldsymbol{GLP}_u}$.

C. Skip Partial Graph Communication Patterns

Next, we generalize Theorem 4 to partial graph communicating patterns. When evaluating a partial GLP's data age, we only consider the partial cause-effect chains formulated by the edges contained in the partial GLP. If the partial cause-effect chain does not include the source task of the complete chain, we consider its data age to be 0.

Example 11. Consider a cause-effect chain $\tau_0 \rightarrow \tau_1 \rightarrow \tau_2$ and two partial GLPs $\boldsymbol{pGLP}_1 = \{\boldsymbol{ELP}_{E_0}(1)\}$ and $\boldsymbol{pGLP}_2 = \{\boldsymbol{ELP}_{E_1}(2)\}$, then the data age evaluation of $\boldsymbol{pGLP}_1$ only considers the cause-effect chain $\tau_0 \rightarrow \tau_1$. The data age of $\boldsymbol{pGLP}_2$ is 0 because it does not contain the source task τ_0.

Theorem 8. *Theorem 4 applies to partial graph communication patterns.*

Proof. Skipped due to being similar to Theorem 4. ■

Definition VII.3 (Contain, $\subset$)**.** *Consider two partial GLPs $\boldsymbol{pGLP}_u$ and $\boldsymbol{pGLP}_w$. We say $\boldsymbol{pGLP}_w$ contains $\boldsymbol{pGLP}_u$, denoted as $\boldsymbol{pGLP}_u \subset \boldsymbol{pGLP}_w$, if:*

$$\forall \boldsymbol{pGLP}_u(E_k) \in \boldsymbol{pGLP}_u,\ \boldsymbol{pGLP}_u(E_k) = \boldsymbol{pGLP}_w(E_k)$$

Example 12. Consider three partial graph last-reading patterns $\boldsymbol{pGLP}_u = \{\boldsymbol{ELP}_{E_0}(0)\}$, $\boldsymbol{pGLP}_w = \{\boldsymbol{ELP}_{E_0}(0), \boldsymbol{ELP}_{E_1}(1)\}$ and $\boldsymbol{pGLP}_k = \{\boldsymbol{ELP}_{E_0}(1)\}$. Then $\boldsymbol{pGLP}_w$ contains $\boldsymbol{pGLP}_u$, but does not contain $\boldsymbol{pGLP}_k$.

Theorem 9. *Consider two sub-chains $\mathcal{C}_1 = \tau_0 \to ... \to \tau_k$ and $\mathcal{C}_2 = \tau_{k+1} \to ... \to \tau_n$ of a cause-effect chain $\mathcal{C} = \mathcal{C}_1 \to \mathcal{C}_2$. Given the same task set schedule, the worst-case data age of $\mathcal{C}_1$ is no larger than $\mathcal{C}$.*

Proof. This theorem can be proved following the compositional theorem and its generalization in Günzel *et al.* [18]. ∎

Theorem 10. *Consider two partial GLPs $\boldsymbol{pGLP}_u \subset \boldsymbol{pGLP}_w$, the maximum data age evaluated in $\boldsymbol{pGLP}_w$ (following Definition V.9) is no less than $\boldsymbol{pGLP}_u$.*

Proof. Direct results of Theorem 9 because the cause-effect chains in $\boldsymbol{pGLP}_w$ have more tasks. ∎

D. DART-specific Symbolic Optimization Algorithm

The theorems above can be summarized as follows:

Corollary 1. *Consider an optimization problem that minimizes data age. A partial GLP $\boldsymbol{pGLP}_i$ and all the other partial GLPs that contain $\boldsymbol{pGLP}_i$ can be skipped if one of the evaluated partial GLP $\boldsymbol{pGLP}_j$ satisfies (theorems 8 to 10):*

- *$\boldsymbol{pGLP}_j$'s evaluation is smaller than $\boldsymbol{pGLP}_i$;*
- *$\boldsymbol{pGLP}_i$ is smaller than $\boldsymbol{pGLP}_j$ (Definition VII.2);*

ELP Enumeration Order. ELPs need to be ordered appropriately to fully utilize the symbolic operations. When selecting edges to add to GLPs, select the edges that are closer to the source tasks first. After selecting an edge, first add its largest ELP (follow Definition VII.1) into the GLP when minimizing data age (smallest EFP first for reaction time optimization).

Pseudocode. The pseudocode for data age minimization problem is shown in Algorithm 1. The vector **best_yet_obj** denotes the worst-case data age of each cause-effect chain evaluated by the best-known GLP. In Line 10, the implementation of the function **GetBestPossibleObj** follows Definition V.9 except only considering the sub-chains contained in $\mathbf{P}_{ite}$. The $\geq$ operator in Line 11 requires element-wise greater than to hold. In Line 12, **worsePatterns** is implemented as a First-In-First-Out queue, because recently visited GLPs have more common ELPs with the next GLP, increasing the likelihood of triggering the skip conditions in Corollary 1. The capacity size is experimentally set to 50: a larger capacity incurs more computation overhead when a communication pattern cannot be skipped, while a smaller size implies fewer skipped patterns. In Line 19, the function **RemoveWorseELP** removes an unvisited ELP from **ELP_Set** if adding the ELP to $\mathbf{P}_{ite}$ will make $\mathbf{P}_{ite}$ perform worse (evaluated by Corollary 1) than at least one partial GLP in **worse_patterns**. This function significantly speeds up symbolic optimization by allowing the skipping of partial GLPs, eliminating the need to visit and evaluate all partial and complete GLPs that contained the skipped partial GLP.

Algorithm 1: GLPSymbOpt

```
Input: DAG $\mathbf{G} = (\boldsymbol{\tau}, \mathbf{E})$, GLP $\mathbf{P}_{ite} = \{\}$,
       best_yet_obj $= \{Inf, ..., Inf\}$,
       worsePatterns $= \{\}$, unvisited_Edge $E$=$\mathbf{E}(0)$
Output: best_yet_obj
1  if $P_{ite}$ is complete then
2      Solve problem (10) and Update best_yet_obj
3      return
4  end
5  ELP_Set = GetAllELP($E$)
6  while ELP_Set is not empty do
7      $ELP$ = ELP_Set.Pop_front()
8      $\mathbf{P}_{ite}$.Insert($ELP$)
9      if $P_{ite}$ is feasible then
10         obj = GetBestPossibleObj($\mathbf{P}_{ite}$)
11         if obj $\geq$ best_yet_obj then
12             worsePatterns.push_back($\mathbf{P}_{ite}$)
13             if worsePatterns.size() $> 50$ then
14                 worsePatterns.pop_front()
15             end
16         else
17             GLPSymbOpt($\mathbf{G}$, $\mathbf{P}_{ite}$, best_yet_obj,
                 worsePatterns, NextUnvisitedEdge($\mathbf{P}_{ite}$))
18         end
19         ELP_Set.RemoveWorseELP(worsePatterns)
20     end
21     $\mathbf{P}_{ite}$.Erase($ELP$)
22 end
```

Example 13. In Example 1, the edge iteration order is $E_0 \to E_2 \to E_1$ because E_0 and E_2 contain the source tasks in cause-effect chains. The first GLP to evaluate is $\boldsymbol{GLP}_0 = \{\boldsymbol{ELP}_{E_0}(0), \boldsymbol{ELP}_{E_2}(0), \boldsymbol{ELP}_{E_1}(0)\}$, which is infeasible; The second GLP to try is $\boldsymbol{GLP}_1 = \{\boldsymbol{ELP}_{E_0}(0),\ \boldsymbol{ELP}_{E_2}(0), \boldsymbol{ELP}_{E_1}(1)\}$, which is feasible. Therefore, any GLPs smaller than $\boldsymbol{GLP}_1$ will be skipped based on Theorem 6. After that, only 3 GLPs have to be evaluated: $\boldsymbol{GLP}_2 = \{\boldsymbol{ELP}_{E_0}(1), \boldsymbol{ELP}_{E_2}(0), \boldsymbol{ELP}_{E_1}(0)\}$, $\boldsymbol{GLP}_3 = \{\boldsymbol{ELP}_{E_0}(0),\ \boldsymbol{ELP}_{E_2}(1),\ \boldsymbol{ELP}_{E_1}(0)\}$ and $\boldsymbol{GLP}_4 = \{\boldsymbol{ELP}_{E_0}(1),\ \boldsymbol{ELP}_{E_2}(1),\ \boldsymbol{ELP}_{E_1}(0)\}$. However, neither is feasible. As a result, the number of complete GLPs to evaluate decreases from 36 in Example 7 to only 1, although there are 4 infeasible GLPs to skip.

Theorem 11. *The difference between solutions found by Algorithm 1 and the optimal solution to problem 5 is upper-bounded by $\sum_{\mathcal{C} \in \boldsymbol{\mathcal{C}}} B_{\mathcal{C}}$, where $B_{\mathcal{C}}$ is given by Lemma 4.*

Proof. We prove the theorem under data age optimization. Reaction time optimization can be performed similarly. Let's denote the optimal GLP found by Algorithm 1 as $\boldsymbol{GLP}^{Alg1*}$, the optimal GLP to problem 5 as $\boldsymbol{GLP}^*$. Furthermore, we divide all the possible complete GLPs into two sets: $\boldsymbol{S}^{eval}$ and $\boldsymbol{S}^{skip}$, which denote the complete GLPs evaluated during Algorithm 1 and those skipped. We know $\boldsymbol{GLP}^{Alg1*} \in \boldsymbol{S}^{eval}$. If $\boldsymbol{GLP}^* \in \boldsymbol{S}^{eval}$, then Algorithm 1 finds the optimal solutions because Algorithm 1 will select the best GLPs within $\boldsymbol{S}^{eval}$.

If $\boldsymbol{GLP}^* \in \boldsymbol{S}^{skip}$, then $\boldsymbol{GLP}^*$ is skipped either based on Theorem 6 (performance bound is provided by Theorem 7), or based on Line 12 in Algorithm 1 ($\boldsymbol{GLP}^*$ cannot achieve better performance than $\boldsymbol{GLP}^{Alg1*}$). Therefore, the theorem is proved. ■

E. Computation Efficiency

A simple worst-case complexity analysis when optimizing the data age or time disparity is given as follows:

$$O(\mathbf{E}) = \prod_{E \in \boldsymbol{E}} |\mathbf{ELP}_E| \tag{26}$$

where $\boldsymbol{E}$ denotes the edges that appear in the objective function, $|\cdot|$ denotes the size of a set. Reaction time optimization has a similar complexity. The analysis is pessimistic because it is difficult to analyze the improvements brought by the symbolic operation or the back-tracking algorithms, though they can usually bring significant speed-up.

VIII. Generalizations and Limitations

A. Sporadic Tasks Optimization

Our optimization algorithms can work with cause-effect chains that contain sporadic tasks, i.e., tasks released non-periodically. With the help of the compositional theorem proposed in [18], the cause-effect chain can be separated into several sub-chains where some sub-chains do not contain periodic tasks. The sub-chains with only periodic tasks can still be optimized with the algorithm proposed in this paper. However, notice that evaluating and optimizing time disparity for sporadic tasks may not be easily applicable.

B. Other Objective Functions and Schedulability Analysis

The backtracking algorithm in Section VI-B works with many types of objective functions and their combination. If these objective functions can be transformed into linear functions such as DART, then optimality is guaranteed with good runtime speed; otherwise, the algorithm may still be applicable (e.g., time disparity jitter), though without the optimality guarantee. In the latter case, nonlinear optimization algorithms [40], [41] may also be considered.

C. Pessimistic RTA and Second Step Optimization

It is difficult to directly adopt the exact schedulability analysis [34], [35], [42] when optimizing the virtual offset and virtual deadline because of their discrete and nonlinear forms. Therefore, we can only guarantee optimality with the schedulability analysis method used during optimization.

A potential solution is performing a second optimization step to optimize only the virtual offset based on the exact response time. The exact response time only depends on the virtual offset and can usually be easily obtained. Therefore, we can utilize the exact response time to optimize the virtual deadline to improve performance further. A limitation is that it can only find optimal solutions within the current GCP constraints. Therefore, in experiments, we utilize the heuristic proposed in Maia *et al.* [33] for the second optimization step if the objective functions are reaction time or data age; we can optimize the virtual deadline following Definition V.9 for time disparity and jitter optimization.

IX. Experimental Results and Discussion

The optimization framework was implemented in C++ and tested on a computing cluster (AMD EPYC 7702 CPU). The following baseline methods are roughly ordered from least to most effective based on some experiment performance:

- DefLET, the default LET model.
- Martinez18, an offset optimization method [11] for LET.
- Bradatsch16, it shrinks the length of LET interval to the worst-case response time; the offset is 0 [32].
- Implicit, implicit communication following [3].
- Maia23, it uses the smallest relative start time and biggest relative finish time of each task as the LET interval [33]. We did not compare their JLD optimization algorithm because it changes the scheduling algorithm.
- fLET_GCP_Enum, from Section VI-A.
- fLET_GCP_Backtracking_LP, from Section VI-B.
- fLET_GCP_SymbOpt, Algorithm 1.
- fLET_GCP_Extra, from Section VIII-C.

All the involved LP problems are solved by CPLEX [43]. The full experiments can be reproduced following the repository: https:github.com/zephyr06/LET_OPT.

DBP is not included in our comparison, as it is proven to perform worse than implicit communication theoretically [2].

In experiments, the task set is scheduled by the rate-monotonic algorithm. A safe response time analysis (RTA) is used to obtain the response time R_i for task τ_i [44]:

$$R_i = C_i + \sum_{j \in \text{hp}(i)} \lceil \frac{R_i}{T_j} \rceil C_j \tag{27}$$

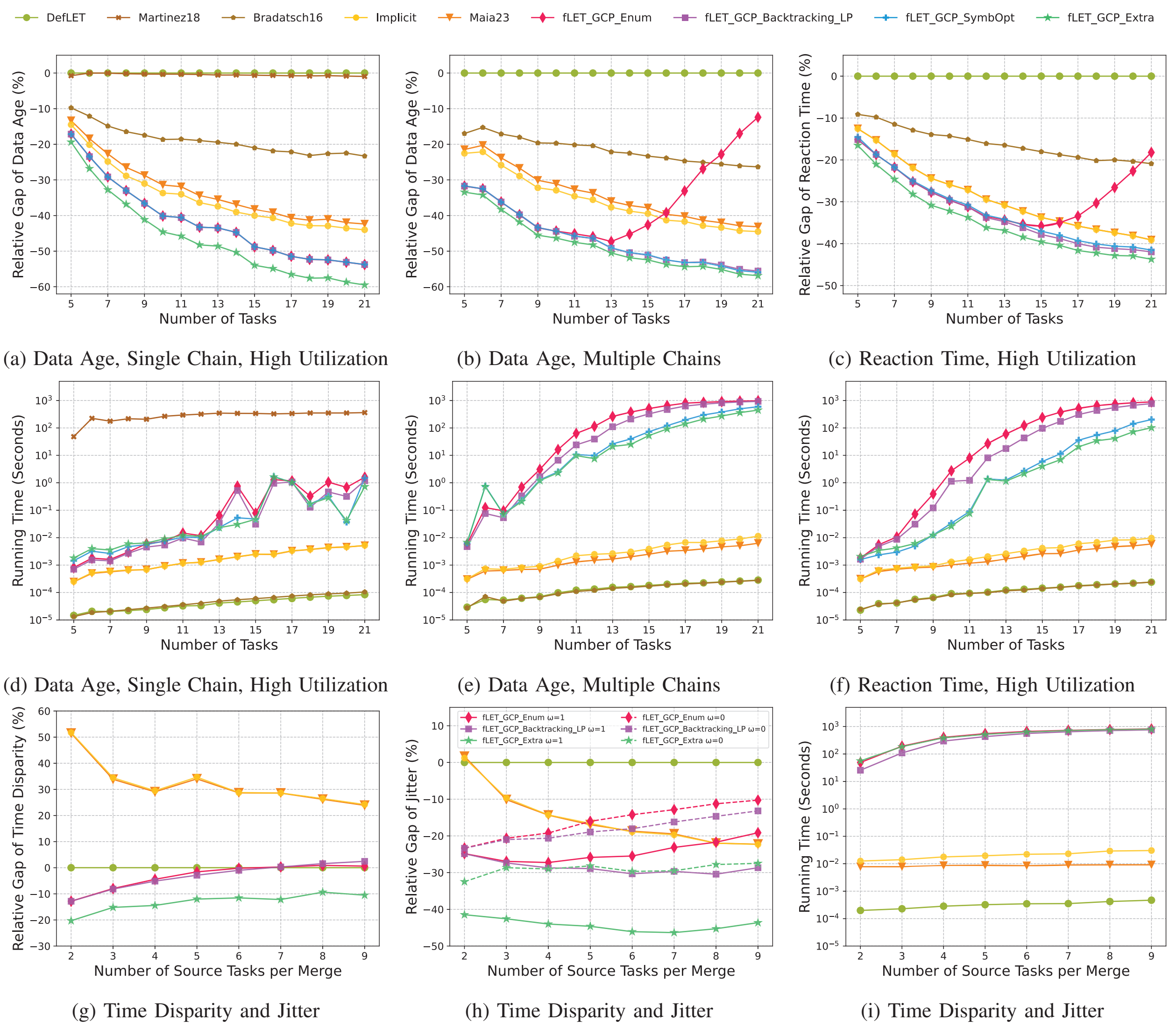
where $\text{hp}(i)$ denotes the tasks with higher priority than τ_i.

The time limit for optimizing one task set is 1000 seconds. If not time out, Martinez18 finds optimal offset assignments; fLET_GCP optimization algorithms find the optimal virtual offset and virtual deadline with respect to the RTA (27).

To maximize the chance of finding a feasible solution within a limited time, the GCP optimization algorithms first evaluate the GCP of the default LET; after that, they follow the searching order mentioned in Section VII-D.

A. Task Set Generation

The DAG task sets are generated following the WATERS Industry Challenge [45]. Task periods are randomly generated from a predefined set $\{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ whose relative probability distribution {3, 2, 2, 25, 25, 3, 20, 1, 4} [45]. The task set utilization is randomly selected from $[0.5, 0.9] \times m$, where m is the number of cores ($m = 4$



(a) Data Age, Single Chain, High Utilization

(b) Data Age, Multiple Chains

(c) Reaction Time, High Utilization

(d) Data Age, Single Chain, High Utilization

(e) Data Age, Multiple Chains

(f) Reaction Time, High Utilization

(g) Time Disparity and Jitter

(h) Time Disparity and Jitter

(i) Time Disparity and Jitter

Figure 5: Relative performance gap and runtime (log scale) when optimizing different performance metrics individually. fLET outperforms other communication protocols and state-of-the-art optimization algorithms while offering broader applicability.

in our case). In high-utilization task sets, the utilization is always $0.9m$. Each task τ_i's execution time C_i is generated by Uunifast [46] except that each task's utilization is no larger than 100%. The deadline D_i^{org} is set as the period T_i.

The DAG structure is generated following He *et al.* [47]. Random edges are added from one task to another with 0.9 probabilities (task sets with many cause-effect chains cannot be generated with smaller probabilities). After generating the DAG, we randomly select source and sink nodes and use the shortest path algorithm by Boost Graph Library [48] to generate random cause-effect chains. The length and the activation pattern distributions of the cause-effect chains follow Table VI and Table VII in Kramer *et al.* [45]. To generate many random task sets while simultaneously satisfying the two distribution requirements, we first generate many random task sets with random cause-effect chains, calculate the likelihood for each task set, and then sample 1000 random task sets weighted by the likelihood for each N. The task sets used in our experiment follow a similar distribution pattern as in Kramer *et al.* [45].

We generate 1000 random task sets for a given number of tasks within a task set. Each task set $\boldsymbol{\tau}$ of N tasks has $1.5N$ to $3N$ random cause-effect chains. For example, there are 31 to 63 cause-effect chains when there are 21 tasks in the task set (following Hamann *et al.* [49]). All the generated task sets are schedulable based on RM.

There are always 21 tasks in each task set when optimizing the time disparity and jitter. However, the maximum number of source tasks varies from 2 to 9 following ROS [25]. Besides, we randomly select 1 to 4 merges to constitute the objective function (6). The weight in the objective function (6) is 1.

Table III: Autonomous robot case study results

	DefLET	Martinez18	Bradatsch16	Implicit	Maia23	fLET_Enum	fLET_Backtracking	fLET_SymbOpt
Reaction Time (ms)	4040	N/A	3237	3237	3237	**2725**	**2725**	**2725**
Data Age (ms)	5000	5000	4197	4197	4197	**3685**	**3685**	**3685**
Sensor Fusion, Jitter (ms)	1500, 1500	N/A	N/A	1712, 1500	N/A	**1461, 1422**	**1461, 1422**	**1461, 1422**

Since finding optimal time disparity jitter is computationally expensive (Observation 1), we optimize only time disparity when evaluating a GLP (Definition V.9) for better efficiency in experiments. We then select the GLP with minimum time disparity and weighted jitter as the final solution. Therefore, optimality for the objective function (6) is not guaranteed, though it is possible by solving mixed-integer programming.

Fig. 5 shows the runtime speed and the performance gap of a method against the default LET, which is defined as

$$\frac{\mathcal{F}_{method} - \mathcal{F}_{defLET}}{\mathcal{F}_{defLET}} \times 100\% \tag{28}$$

Reaction time and data age optimization results are very similar, so we only show one of them under the same situation.

B. Autonomous Robot Case Study

We performed a realistic case study following the autonomous robot system built in Sifat *et al.* [50]. The computation tasks and their dependency relationship are shown in Fig. 6. The tasks are executed in a real embedded system (NVIDIA Jetson AGX Xavier), and their WCET and period are shown in Table. IV. The tasks adopt implicit deadlines. The end-to-end latency is measured for the critical cause-effect chain: SLAM $\rightarrow$ Path Planning $\rightarrow$ Control. The original paper [50] does not consider time disparity, so we selected a reasonable merge (Depth Estimation and Path Planning to Control) to conduct experiments on time disparity and jitter. The results are shown in Table. III.

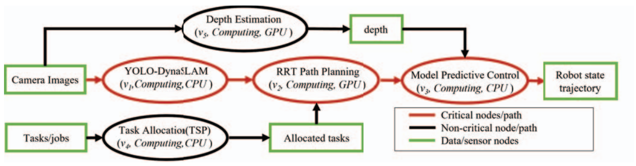

Depth Estimation (v_5, Computing, GPU)
depth
Camera Images
YOLO-DynaSLAM (v_1,Computing,CPU)
RRT Path Planning (v_2, Computing, GPU)
Model Predictive Control (v_3, Computing, CPU)
Robot state trajectory
Tasks/jobs
Task Allocation(TSP) (v_4, Computing, CPU)
Allocated tasks
Critical nodes/path
Non-critical node/path
Data/sensor nodes

Figure 6: Autonomous robot tasks' dependency graph, [50].

Table IV: Autonomous robot computing tasks

Task	Period (ms)	WCET (ms)
SLAM	1000	500
Path Planning	2000	1188
Control	40	37
Task Allocation	10000	10000
Depth Estimation	500	400

C. Performance Analysis

1) fLET vs Other Communication Mechanisms: The performance improvements and the extra time determinism benefits make fLET more competitive than other communication mechanisms, such as default LET and implicit communication, especially when the end-to-end latency metrics are important. Furthermore, fLET maintains the extra time determinism benefits that implicit communication does not have.

2) fLET Optimization Algorithms: In experiments, the suboptimality gap in Theorem 11 is typically under 1%. Meanwhile, the symbolic operation brings 2x to 30x speed-up compared with the backtracking algorithm. The speed advantages are crucial for large task sets or situations with tight time budgets because the algorithm performance may otherwise seriously degrade (e.g., Martinez18 and fLET_GCP_Enum).

3) Multi-objective optimization: In Fig. 5g, the time disparity results nearly overlap between $w = 0$ and $w = 1$, so we only show the results for $w = 1$. Fig. 5g and 5h show that the fLET optimization algorithms effectively balance conflicting goals (time disparity and jitter). Although they cannot guarantee optimality when jitter is present in the objective functions, fLET still significantly outperforms the baseline methods. It is also interesting to note that optimizing the data age often inherently optimizes the reaction time and vice versa. This observation is consistent with the recent work [24], which proves that the maximum reaction time is equivalent to the maximum data age (note that they use a slightly different definition than those introduced in Section III).

4) Pessimistic vs exact RTA: Performing extra optimization with the exact RTA brings big performance improvements when optimizing time disparity and jitter (TDJ). In contrast, the improvements when optimizing DART are more limited. This is probably because TDJ is more "nonlinear" than DART (Theorem 1) and more challenging to optimize.

5) Time-out issue: Despite the possibility for timeouts as systems scale, Fig. 5 shows that substantial performance improvements are still attainable even if the fLET optimization algorithms did not finish within the time limits.

X. Conclusions and Future Work

In this paper, we propose novel optimization algorithms to optimize both the reading and writing time of the LET model, improving the system performance with dataflow determinism preserved. In particular, the optimization problem is reformulated to optimize communication patterns, which can be solved efficiently with LP while supporting the optimization of many metrics, such as end-to-end latency, time disparity, and its jitter. We also introduce novel symbolic operations and prove bounded suboptimality when minimizing the worst-case data age or reaction time. Experimental results highlight significant performance improvement over other communication mechanisms (implicit communication and DBP) and state-of-the-art LET extensions.

XI. Acknowledgment

This work is partially supported by NSF Grants No. 1812963 and 1932074. We also sincerely appreciate the anonymous reviewers and the shepherd whose insightful feedback and guidance significantly contributed to the refinement and enhancement of this work.

References

[1] T. A. Henzinger, B. Horowitz, and C. M. Kirsch, "Giotto: a time-triggered language for embedded programming," in *Proceedings of the IEEE*, 2001.

[2] Y. Tang, X. Jiang, N. Guan, D. Ji, X. Luo, and W. Yi, "Comparing communication paradigms in cause-effect chains," *IEEE Transactions on Computers*, vol. 72, pp. 82–96, 2023.

[3] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, "Communication centric design in complex automotive embedded systems," in *Euromicro Conference on Real-Time Systems*, 2017.

[4] P. Pazzaglia, D. Casini, A. Biondi, and M. D. Natale, "Optimizing inter-core communications under the let paradigm using dma engines," *IEEE Transactions on Computers*, vol. 72, pp. 127–139, 2023.

[5] P. Pazzaglia, D. Casini, A. Biondi, and M. D. Natale, "Optimal memory allocation and scheduling for dma data transfers under the let paradigm," *58th ACM/IEEE Design Automation Conference*, pp. 1171–1176, 2021.

[6] P. Pazzaglia, A. Biondi, and M. D. Natale, "Optimizing the functional deployment on multicore platforms with logical execution time," *IEEE Real-Time Systems Symposium*, pp. 207–219, 2019.

[7] A. Biondi and M. D. Natale, "Achieving predictable multicore execution of automotive applications using the let paradigm," *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 240–250, 2018.

[8] A. M. Kordon and N. Tang, "Evaluation of the age latency of a real-time communicating system using the let paradigm," in *Euromicro Conference on Real-Time Systems*, 2020.

[9] A. Shrivastava and P. Derler, "Introduction to the special issue on time for cps (tcps)," *ACM Transactions on Cyber-Physical Systems*, vol. 5, pp. 1 – 2, 2021.

[10] D. Ziegenbein and A. Hamann, "Timing-aware control software design for automotive systems," in *ACM/IEEE Design Automation Conference*, pp. 56:1–56:6, 2015.

[11] J. Martinez, I. Sañudo, and M. Bertogna, "Analytical characterization of end-to-end communication delays with logical execution time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, pp. 2244–2254, 2018.

[12] PerceptIn, "2021 rtss industry challenge." http://2021.rtss.org/industry-session/, 2021.

[13] C. M. Kirsch and R. Sengupta, "The evolution of real-time programming," in *Handbook of Real-Time and Embedded Systems*, pp. 11–1, Figure 1.12, 2006.

[14] E. Bini, P. Pazzaglia, and M. Maggio, "Zero-jitter chains of periodic let tasks via algebraic rings," *IEEE Transactions on Computers*, vol. 72, no. 11, pp. 3057–3071, 2023.

[15] AUTOSAR, "Autosar timing extensions document," 2022-11-24.

[16] N. Feiertag, K. Richter, J. E. Nordlander, and J. Å. Jönsson, "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics," in *RTSS 2009*, 2008.

[17] J. Abdullah, G. Dai, and W. Yi, "Worst-case cause-effect reaction latency in systems with non-blocking communication," *Design, Automation & Test in Europe*, pp. 1625–1630, 2019.

[18] 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," *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 40–52, 2021.

[19] M. Dürr, G. von der Brüggen, K.-H. Chen, and J.-J. Chen, "End-to-end timing analysis of sporadic cause-effect chains in distributed systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, pp. 1 – 24, 2019.

[20] J. Schlatow, M. Möstl, S. Tobuschat, T. Ishigooka, and R. Ernst, "Data-age analysis and optimisation for cause-effect chains in automotive control systems," *IEEE Symposium on Industrial Embedded Systems*, pp. 1–9, 2018.

[21] T. Kloda, A. Bertout, and Y. Sorel, "Latency analysis for data chains of real-time periodic tasks," *IEEE Conference on Emerging Technologies and Factory Automation*, pp. 360–367, 2018.

[22] M. Verucchi, M. Theile, M. Caccamo, and M. Bertogna, "Latency-aware generation of single-rate dags from multi-rate task sets," *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 226–238, 2020.

[23] T. Klaus, M. Becker, W. Schröder-Preikschat, and P. Ulbrich, "Constrained data-age with job-level dependencies: How to reconcile tight bounds and overheads," *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 66–79, 2021.

[24] M. Günzel, H. Teper, K.-H. Chen, G. von der Brüggen, and J.-J. Chen, "On the equivalence of maximum reaction time and maximum data age for cause-effect chains," in *Euromicro Conference on Real-Time Systems*, 2023.

[25] R. Li, N. Guan, X. Jiang, Z. Guo, Z. Dong, and M. Lv, "Worst-case time disparity analysis of message synchronization in ros," *IEEE Real-Time Systems Symposium*, pp. 40–52, 2022.

[26] X. Jiang, X. Luo, N. Guan, Z. Dong, S. Liu, and W. Yi, "Analysis and optimization of worst-case time disparity in cause-effect chains," *Design, Automation & Test in Europe*, pp. 1–6, 2023.

[27] R. Ernst, S. Kuntz, S. Quinton, and M. Simons, "The logical execution time paradigm: New perspectives for multicore systems (dagstuhl seminar 18092)," *Dagstuhl Reports*, vol. 8, pp. 122–149, 2018.

[28] K.-B. Gemlau, L. KÖHLER, R. Ernst, and S. Quinton, "System-level logical execution time: Augmenting the logical execution time paradigm for distributed real-time automotive software," *ACM Trans. Cyber-Phys. Syst.*, vol. 5, jan 2021.

[29] J. Martinez, I. Sañudo, and M. Bertogna, "End-to-end latency characterization of task communication models for automotive systems," *Real-Time Systems*, pp. 1–33, 2020.

[30] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "End-to-end timing analysis of cause-effect chains in automotive embedded systems," *J. Syst. Archit.*, vol. 80, pp. 104–113, 2017.

[31] E. A. Lee and M. Lohstroh, "Generalizing logical execution time," in *Principles of Systems Design*, 2022.

[32] C. Bradatsch, F. Kluge, and T. Ungerer, "Data age diminution in the logical execution time model," in *ARCS*, 2016.

[33] L. Maia and G. Fohler, "Reducing end-to-end latencies of multi-rate cause-effect chains for the let model," *ArXiv*, vol. abs/2305.02121, 2023.

[34] K. Tindell, "Adding time-offsets to schedulability analysis," 1994.

[35] J. C. Palencia and M. G. Harbour, "Schedulability analysis for tasks with static and dynamic offsets," *IEEE Real-Time Systems Symposium*, pp. 26–37, 1998.

[36] M. Günzel, K.-H. Chen, N. Ueter, G. v. der Brüggen, M. Dürr, and J.-J. Chen, "Compositional timing analysis of asynchronized distributed cause-effect chains," *ACM Trans. Embed. Comput. Syst.*, mar 2023. Just Accepted.

[37] C. Sofronis, S. Tripakis, and P. Caspi, "A memory-optimal buffering protocol for preservation of synchronous semantics under preemptive scheduling," in *Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pp. 21–33, 2006.

[38] D.-S. Chen, R. G. Batson, and Y. Dang, "Applied integer programming: Modeling and solution," 2010.

[39] Wikipedia contributors, "Multigraph — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Multigraph&oldid=1158564961, 2023. [Online; accessed 25-October-2023].

[40] S. Wang, R. K. Williams, and H. Zeng, "A general and scalable method for optimizing real-time systems with continuous variables," *IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 119–132, 2023.

[41] S. Wang, D. Li, S.-Y. Huang, X. Deng, A. H. Sifat, C. Jung, R. Williams, and H. Zeng, "A general and scalable method for optimizing real-time systems," *ArXiv*, vol. abs/2401.03284, 2024.

[42] O. Redell and M. Törngren, "Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter," *Proceedings. Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 164–172, 2002.

[43] IBM ILOG, "V12. 1: User's manual for cplex," *International Business Machines Corporation*, vol. 46, no. 53, p. 157, 2009.

[44] M. Joseph and P. K. Pandya, "Finding response times in a real-time system," *Comput. J.*, vol. 29, pp. 390–395, 1986.

[45] A. H. Simon Kramer, Dirk Ziegenbein, “Real world automotive benchmarks for free,” in *Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2015.

[46] E. Bini and G. Buttazzo, “Measuring the performance of schedulability tests,” *Real-Time Systems*, vol. 30, pp. 129–154, 2005.

[47] Q. He, M. Lv, and N. Guan, “Response time bounds for dag tasks with arbitrary intra-task priority assignment,” in *ECRTS*, 2021.

[48] J. G. Siek, L.-Q. Lee, and A. Lumsdaine, “The boost graph library - user guide and reference manual,” in *C++ in-depth series*, 2001.

[49] A. Hamann, S. Kramer, M. Pressler, D. Dasari, F. Wurst, and D. Ziegenbein, “Waters industrial challenge 2017,” in *Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2017.

[50] A. H. Sifat, X. Deng, B. Bharmal, S. Wang, S.-Y. Huang, J. Huang, C. Jung, H. Zeng, and R. K. Williams, “A safety-performance metric enabling computational awareness in autonomous robots,” *IEEE Robotics and Automation Letters*, vol. 8, pp. 5727–5734, 2023.