Efficient Coordination for Distributed Discrete-Event Systems

Byeonggil Jun
Arizona State university
byeonggil@asu.edu

Edward A. Lee
University of California, Berkeley
eal@berkeley.edu

Marten Lohstroh
University of California, Berkeley
marten@berkeley.edu

Hokeun Kim
Arizona State university
hokeun@asu.edu

***Abstract*—Timing control while preserving determinism is often a key requirement for ensuring the safety and correctness of distributed cyber-physical systems (CPS). Discrete-event (DE) systems provide a suitable model of computation (MoC) for time-sensitive distributed CPS. The high-level architecture (HLA) is a useful tool for the distributed *simulation* of DE systems, but its techniques can be adapted for *implementing* distributed CPS. However, HLA incurs considerable overhead in network messages conveying timing information between the distributed nodes and the centralized run-time infrastructure (RTI). This paper gives a novel approach and implementation that reduces such network messages while preserving DE semantics. An evaluation of our runtime demonstrates that our approach significantly reduces the volume of messages for timing information in HLA.**

***Index Terms*—High level architecture, Cyber-physical systems, Distributed systems, Discrete-event systems, Real-time systems**

I. Introduction

Distributed cyber-physical systems (CPS) interacting with the physical world and connected over the networks are becoming more pervasive and widely used. A distributed CPS often requires timing control over the network with determinism, ensuring the same outputs and behavior for given initial conditions and inputs. One of the ways to support such determinism in distributed CPS is the high-level architecture (HLA) [1], an IEEE standard for discrete event (DE) system simulation. The methods of HLA have also been extended to support features for system implementation (vs. simulation).

An HLA-based system includes a run-time infrastructure (RTI) such as CERTI [2], a centralized coordinator, and federates, distributed individual nodes. During the simulation or execution of an HLA-based system, federates react to events in a logical time order. Events have a timestamp on the globally agreed logical timeline [3]. To ensure that federates see events in logical time order, federates exchange signals that include timing information with the RTI. However, the frequent exchange of such signals drastically increases network overhead, potentially causing issues, as shown in our case study of implementing distributed CPS using HLA [4].

This paper introduces methods that significantly reduce the number of signals in HLA-based distributed systems. We use an open-source coordination language, Lingua Franca (LF) [5], as our baseline because LF provides an extended, working implementation with mechanisms similar to HLA. The centralized coordinator of LF is loosely based on HLA [6]. We examine the purpose of each signal that is used for coordinating logical time and find scenarios where signals are used inefficiently. Then, we provide solutions to eliminate the signals that are not essential for ensuring determinism in HLA.

II. Related Work

A number of algorithms and methods for synchronization mechanisms have been proposed for distributed discrete event (DDE) simulation [7]. Among those, HLA [1] has been widely used and standardized by the IEEE.

There has been research on optimizing the performance of DDE systems and HLA with RTI. Rudie *et al.* [8] present a strategy to minimize the communication in DDE systems by producing minimal sets of communications. Wang and Turner [9] propose an optimistic time synchronization of HLA using an RTI with a rollback ability when events turn out to be incorrectly scheduled during the optimistic simulation, although such approaches are not for deployment where a rollback is impossible. COSSIM [10] provides time synchronization that can trade off the timing accuracy and performance with relaxed synchronization for CPS simulation using an IEEE HLA-compliant interface. Distinct from COSSIM, our approach improves efficiency while strictly synchronizing a DDE system with an enhancement to standard HLA.

III. Background

Lingua Franca (LF) is an open-source coordination language and runtime implementing reactors [11], [12]. Reactors adopt advantageous semantic features from established models of computation, particularly actors [13], logical execution time [14], synchronous reactive languages [15], and discrete event systems [16]. LF also enables deterministic interactions between physical and logical timelines [5].

LF adopts the superdense model of time for logical time [5]. Each tag $g \in \mathbb{G}$ is a pair of a *time value* $t \in \mathbb{T}$ and a *microstep* $m \in \mathbb{N}$. The time value t includes limiting values, *NEVER*, a time value earlier than any other, and *FOREVER*, a time value larger than any other, represented by symbols $-\infty$ and ∞ in this paper, respectively.

Below are formal set definitions of $\mathbb{N}, \mathbb{T}$ and $\mathbb{G}$ for our discussion in this paper:

- $\mathbb{N}$: A set of non-negative integers that can be represented by a 32-bit unsigned integer.
- $\mathbb{T}$: A set of non-negative integers that can be represented by a 64-bit signed integer $\cup \{-\infty\}$.
- $\mathbb{G}$: A set defined as $\{(t, m) \mid t \in \mathbb{T} \setminus \{-\infty, \infty\},\ m \in \mathbb{N}\} \cup \{(-\infty, 0), (\infty, M_{\max})\}$.

The set of tags, $\mathbb{G}$, forms a totally ordered set where, given two tags $g_a = (t_a, m_a)$ and $g_b = (t_b, m_b)$, $g_a < g_b$ if and only if 1) $t_a < t_a$ or 2) $t_a = t_a$ and $m_b < m_b$

2832-6520/24/$31.00 ©2024 IEEE
DOI 10.1109/MEMOCODE63347.2024.00019

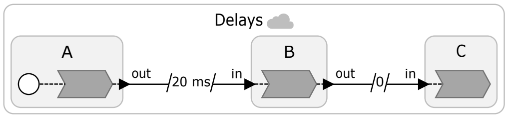

Fig. 1: An example LF program with delays.

Name	Payload	Description	Direction
MSG_{ij}	Tag & Message	Tagged Message	j to i via RTI
LTC_j	Tag	Latest Tag Complete	j to RTI
NET_j	Tag	Next Event Tag	j to RTI
TAG_i	Tag	Tag Advance Grant	RTI to i

TABLE I: Types of messages and signals related to time management that are exchanged by federates and the RTI.

We introduce a function A, an operation like tag addition. Here, we handle overflow in the formalization, recognizing that useful programs should not encounter overflow; We formally define $A\colon \mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}$ as shown below:

$$A(g_a, g_b) = A((t_a, m_a), (t_b, m_b)) =$$

$$\begin{cases} (t_a, m_a + m_b), & \text{if } 0 \le t_a < \infty \wedge t_b = 0 \wedge m_a + m_b < M_{\text{max}} \\ (t_a + t_b, m_b), & \text{if } 0 \le t_a < \infty \wedge t_b > 0 \wedge t_a + t_b < \infty \\ (t_a, M_{\text{max}}), & \text{if } 0 \le t_a < \infty \wedge t_b = 0 \wedge m_a + m_b \ge M_{\text{max}} \\ (\infty,\ M_{\text{max}}), & \text{if } 0 \le t_a < \infty \wedge t_b > 0 \wedge t_a + t_b \ge \infty \\ (-\infty,\ 0), & \text{if } t_a = -\infty \vee t_b - \infty \\ (\infty,\ M_{\text{max}}) & \text{if } t_a = \infty \end{cases} \tag{1}$$

Note that if t_b is greater than 0, $A((t_a, m_a), (t_b, m_b))$ yields a tag that has m_b as the microstep, effectively ignoring m_a.

For simplicity, from now on, we will use an *elapsed* tag, the elapsed time and microstep since the startup of the program, instead of an *actual* tag with the time from January 1, 1970. Formally, when an actual tag is $g = (t, m)$ and the startup time of the program t_s, then the elapsed tag is $(t - t_s, m)$.

Fig. 1 shows a diagram of an LF program named *Delays* with three reactor instances, a, b, and c, which are instances of reactor classes A, B, and C. The cloud symbol indicates that this is a federated program where each top-level reactor becomes a federate that can be deployed to remote machines. Dark gray chevrons indicate reactions that are executed when triggered. The white circle in a denotes a startup trigger.

LF supports the concept of *logical delay* to explicitly represent logical time elapsing through a connection. As after delays in LF are specified using time values, to make use of function A, we need to convert the delay represented by a time value to a tag. We introduce a function $C\colon \mathbb{T} \longrightarrow \mathbb{G}$ for converting a time value of delay to a tag:

$$C(d) = \begin{cases} (0,\ 1), & \text{if } d = 0 \\ (d,\ 0), & \text{if } 0 < d < \infty \\ (0,\ 0), & \text{if } d = -\infty \\ (\infty,\ M_{\text{max}}) & \text{if } d = \infty \end{cases} \tag{2}$$

Specifically, when a reactor sends a message with tag $g_s = (t_s, m_s)$ through a connection with logical delay $d \in \mathbb{T}$, the resulting tag g_d is $A(g_s, C(d))$. Let D_{ij} be the *minimum tag increment delay* over all connections from j to i. This means that if a federate j sends a message with a tag g_j, this may cause a message for i with a tag $A(g_j, D_{ij})$, but no earlier.

A federate can advance its logical time to a tag g only if the RTI guarantees that the federate will not later receive any messages with tags earlier than or equal to g. The RTI and federates continuously exchange the signals LTC, NET, and TAG that are briefly described in TABLE I to keep track of each federate's state and manage time advancement. We use a function G to denote the payload tag of a signal or a message. For example, $G(\text{MSG}_{ij})$ is the tag of the message MSG_{ij}. When federate i receives MSG_{ij}, it schedules an event at $G(\text{MSG}_{ij})$ to process the message.

LTC_j (Latest Tag Complete) is sent from a federate j to the RTI to notify that federate j has finished $G(\text{LTC}_j)$. If there is a connection from federate j to i.

NET_j (Next Event Tag) is sent from federate j to the RTI to report the tag of the earliest unprocessed event of j. This signal promises that j will not later produce any messages with tags earlier than $G(\text{NET}_j)$ unless it receives a new message from the network with a tag earlier than $G(\text{NET}_j)$.

TAG_i (Tag Advance Grant) is sent by the RTI to federate i. When i receives this signal, it knows it has received every message with a tag less than or equal to $G(\text{TAG}_i)$. It can now advance its tag to $G(\text{TAG}_i)$ and process all events with tags earlier than or equal to $G(\text{TAG}_i)$. If a federate i has no upstream federate, then i can advance its tag without TAG.

For each federate j, the RTI maintains variables N_j and L_j and a priority queue Q_j called the *in-transit message queue* to predict j's future behavior. N_j is the tag of the latest received NET_j. L_j is the tag of the latest received LTC_j. Q_j is a priority queue that stores tags of in-flight messages that have been sent to j, sorted by tag. When the RTI forwards a message to j, it stores the tag in Q_j, and when the RTI receives LTC_j, it removes tags earlier than or equal to $G(\text{LTC}_j)$ from Q_j. Let $H(Q_j)$ denote the head of Q_j.

When the RTI receives NET_i, it updates N_i and decides whether to send TAG_i. Let U_i denote the set of federates immediately upstream of i (those with direct connections). Let $g = \min_{j \in U_i} D(L_j, D_{ij})$. If $g \ge N_i$, then the RTI grants TAG_i with a tag g, allowing it to process its events. If $g < N_i$, it may still be possible to grant a TAG_i by computing B_i, the **earliest (future) incoming message tag** for node i. The RTI can send TAG_i with a tag $\min(N_i, H(Q_i))$ if $B_i > \min(N_i, H(Q_i))$.

To calculate B_i, consider an immediate upstream federate $j \in U_i$. The earliest possible tag of a future message from j that the RTI might see is $\min(B_j, N_j, H(Q_j))$. Consequently, we can compute B_i recursively as:

$$B_i = \min_{j \in U_i}(A(\min(B_j, N_j, H(Q_j)), D_{ij}))) \tag{3}$$

The RTI can easily calculate this quantity unless there is a cycle (paths from i back to itself) with no after delays [17]. In this paper, we simply ignore federates within zero-delay cycles and do not apply our optimizations to them.

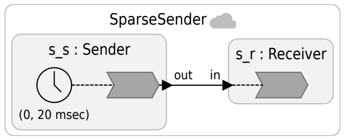

Fig. 2: An LF program where the federate Sender sends messages sparsely.

IV. Inefficiency In Time Management Protocol

In this section, we discuss inefficiencies in LF's HLA-based timing coordination. As described in Section III, a federate sends NET every time it completes a tag, which is unnecessary.

Federate i sends NET_i for two reasons: (1) to notify i's next event tag to the RTI to let the RTI compute TAG signals for i's downstream federates and (2) to request a TAG_i to advance i's tag to $G(\text{NET}_i)$. Therefore, a NET_i signal is unnecessary if it results in no TAG signal for downstream federates and if i can safely advance to a tag $G(\text{NET}_i)$.

We show how unnecessary NET signals are produced using a simple LF example shown in Fig. 2. The timer in the upstream federate s_s triggers s_s's reaction every 20 ms. Assume s_s is a federate polling a distance sensor and the downstream federate s_r processes the sensing result. For instance, the sensor can be used for detecting an emergency situation in an autonomous vehicle or detecting cars entering and exiting a parking lot [4]. The sensor only occasionally detects an interesting event, so it sends a message sparsely.

Fig. 3 shows the trace of the first 100 ms of execution of Fig. 2. The trace does not show the signals at startup tag $(0,\ 0)$ that are irrelevant to our discussion. Federate s_s sends a tagged message every 100 ms in this program. After the startup tag, federate s_r sends $\text{NET}_{s\_r}((\infty,\ M_{\max}))$ in ①, which indicates that it has no event to execute, while s_s sends NET signals every 20 ms. Most of the NET signals from s_s are unnecessary because s_s can advance its tag without TAG signals and the RTI cannot grant $\text{TAG}_{s\_r}((\infty,\ M_{\max}))$ to s_r with those NET signals. At $(100\ ms,\ 0)$, s_s sends a tagged message via the RTI and the RTI knows that $\min(N_{s\_r}, H(Q_{s\_r}))$ is $(100\ ms,\ 0)$. So the RTI sends $\text{TAG}_{s\_r}((100\ ms,\ 0))$ in ④ based on $\text{LTC}_{s\_s}((100\ ms,\ 0))$ and $\text{NET}_{s\_r}((120\ ms,\ 0))$, signals ② and ③, respectively. Thus, $\text{NET}((120\ ms,\ 0))$ is necessary.

V. Downstream Next Event Tag

We introduce a new signal, Downstream Next Event Tag (DNET), to eliminate unnecessary NET signals. The RTI sends DNET_j to federate j with the tag $G(\text{DNET}_j)$ when all NET_j signals such that $G(\text{NET}_j) \leq G(\text{DNET}_j)$ are not necessary for computing TAG for j's downstream federates. Then, j does not have to send NET_j when j's next event tag is earlier than or equal to $G(\text{DNET}_j)$, and j can advance its logical time. Note that j still must send NET_j signals in case j cannot advance to its next event's tag.

The RTI has to calculate $G(\text{DNET}_i)$ carefully to prevent i from skipping sending necessary NET_i while removing every unnecessary NET_i signal. If $G(\text{DNET}_i)$ is too early, i would send unnecessary NET_i, and if $G(\text{DNET}_i)$ is too late, i may not send a necessary NET_i, resulting in i's downstream federates being unable to advance their tags.

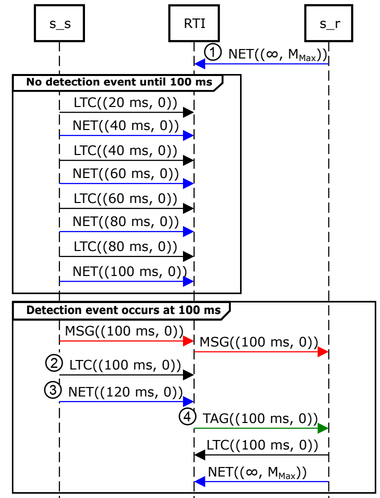

Fig. 3: An execution trace of the program in Fig. 2.

Let $\bar{D}_j$ denote the set of federates "transitive" downstream of federate j, specifically, those that are connected from j via one or more chained connections. For every downstream node i, the RTI tries to grant TAG_i of $\min(N_i, H(Q_i))$. Thus, the RTI has to find an upper bound of the tags of unnecessary NET_j signals. Let g be the latest tag to which j can advance without sending NET_j for a federate $i \in \bar{D}_j$. g must be the latest tag that satisfies the condition:

$$A(g, D_{ij}) <= \min(N_i, D_{ij})$$

For example, if $\min(N_i, H(Q_i))$ is $(2\ s,\ 3)$ and D_{ij} is $(0,\ 3)$, then a tag $(2\ s,\ 0)$ is an upper bound of unnecessary NET_j's tag. Concretely, NET_j with $G(\text{NET}_j) = (2\ s,\ 0)$ is unnecessary because $B_i = A((2\ s,\ 0), (0,\ 3)) = (2\ s,\ 3)$ is not later than $\min(N_i, H(Q_i))$. If, on the other hand, the RTI receives NET_j with $G(\text{NET}_j) = (2\ s,\ 1)$, the earliest tag among tags later than $(2\ s,\ 0)$, it *can* grant a TAG_i $((2\ s,\ 3))$ because $B_i = A((2\ s,\ 1), (0,\ 3)) = (2\ s, 4) > (2\ s,\ 3)$.

To calculate the upper bound of the tags of unnecessary NET_j signals, we define a function S that behaves like tag subtraction. What we really need is subtraction, but because A saturates the microstep on overflow, there is no function S such that if $g = S(g_a, g_b)$ then $A(g, g_b) = A(S(g_a, g_b), g_b) = g_a$, which is what a true subtraction function would do. Instead, we define function S that returns a tag $g = S(g_a, g_b)$ where g is the latest tag of the set of tags that satisfy:

$$A(g, g_b) = A(S(g_a, g_b), g_b) <= g_a \tag{4}$$

Formally, we define the function $S\colon \mathbb{G} \times \mathbb{G}_b \longrightarrow \mathbb{G}$ as:

$$S(g_a, g_b) = S((t_a, m_a), (t_b, m_b)) = \begin{cases} (-\infty,\ 0), & \text{if } g_a = -\infty \vee g_a < g_b \\ (t_a - t_b, m_a - m_b), & \text{if } \infty > t_a \geq t_b = 0 \wedge m_a \geq m_b \\ (t_a - t_b, M_{\max}), & \text{if } \infty > t_a \geq t_b > 0 \wedge m_a \geq m_b \\ (t_a - t_b - 1, M_{\max}), & \text{if } \infty > t_a > t_b > 0 \wedge m_a < m_b \\ (\infty,\ M_{\max}) & \text{if } t_a = \infty \end{cases} \tag{5}$$

where $\mathbb{G}_b$ is $\mathbb{G} \setminus \{(-\infty,\ 0),\ (\infty,\ M_{\max})\}$. When we use function S to compute $G(\text{DNET}_j)$, g_a is $\min(N_i, H(Q_i))$ and g_b is D_{ij}. If $i \in \bar{D}_j$, the value D_{ij} cannot be $(\infty,\ M_{\max})$ because D_{ij} of $(\infty,\ M_{\max})$ means there is no path from j to i. Also, D_{ij} cannot be $(-\infty,\ 0)$ because D_{ij} is always greater than or equal to $(0,\ 0)$ ("no delay" is encoded to $(0,\ 0)$).

We know that the RTI *cannot* send TAG_i with a tag $\min(N_i, H(Q_i))$ if j sends NET_j such that $G(\text{NET}_j) \leq S(\min(N_i, H(Q_i)), D_{ij})$. Thus, we compute $G(\text{DNET}_j)$, the upper bound of the tags of NET_j signals that are unnecessary for j's *every* downstream federate, as:

$$\min_{\forall i \in \bar{D}_j} (S(\min(N_i, H(Q_i)), D_{ij})) \tag{6}$$

When the value of Equation (6) changes due to any update to N_i or $H(Q_i)$ and j has not sent any necessary NET_j signals, the RTI needs to send new DNET_j to j.

Now we describe how federates deal with DNET signals. Each federate j maintains a variable DN_j which stores the most recent DNET_j's tag. At the start of the execution, j initializes DN_j to $(-\infty,\ 0)$. Upon deciding not to send its next event tag, j stores the tag in a variable SN_j, which denotes the last skipped next event tag. When j sends NET_j, it resets SN_j to $(-\infty,\ 0)$. The variable SN_j is needed when a skipped NET signal is revealed to be necessary later.

There are three cases where a federate j must update or use DN_j or SN_j to decide whether to send NET_j signals.

First, when j completes a tag g_j and has a next event at a tag g'_j, j compares DN_j against g'_j. Assume that j can advance to g'_j. 1) If $DN_j < g'_j$, the RTI needs NET_j (g'_j) for at least one of j's downstream federates. j must send $\text{NET}_j(g'_j)$ and reset SN_j to $(-\infty,\ 0)$. 2) If $DN_j \geq g'_j$, none of federate in $\bar{D}_j$ requires $\text{NET}_j(g'_j)$. The federate j decides not to send $\text{NET}_j(g'_j)$. And thus, j stores the tag g'_j to SN_j.

Second, when j receives a new DNET_j, it checks whether $G(\text{DNET}_j)$ is earlier than SN_j. 1) If $G(\text{DNET}_j) < SN_j$, at least one downstream federate is waiting for j's NET_j with a tag later than $G(\text{DNET}_j)$. So j must send NET_j with the tag SN_j and stores $G(\text{DNET}_j)$ in DN_j and resets SN_j to $(-\infty,\ 0)$. 2) If $G(\text{DNET}_j) \geq SN_j$, no federate requires $\text{NET}_j(SN_j)$. j only changes DN_j to the new $G(\text{DNET}_j)$.

Third, when j sends a new tagged message MSG_{ij} to federate i with a destination tag g_t, j compares g_t against DN_j. If 1) If $g_t < DN_j$, j knows that i has an event at g_t. Thus, j updates DN_j to g_t. This allows j to send necessary NET_j without having to wait for a new DNET_j. 2) If $g_t \geq DN_j$, no action is required. j will send NET_j after it completes the current tag anyway.

Fig. 4 shows the trace of the LF program in Fig. 2 where DNET signals are used for removing unnecessary NET signals. When the RTI receives $\text{NET}_{s\_r}((1\ \ s,\ \ 0))$ in ⑤ from s_r, it sends $\text{DNET}_{s\_s}((1\ \ s,\ \ 0))$ in ⑥ to s_s as $S((1\ \ s,\ \ 0),(0,\ \ 0))$ is $(1\ \ s,\ \ 0)$. Consequently, s_s does not send NET signals until $(100\ \ ms,\ \ 0)$. At $(100\ \ ms,\ \ 0)$, s_s sends $\text{MSG}_{s\_rs\_s}((100\ \ ms,\ \ 0))$ in ⑦. Now, the RTI knows that s_r has an event at $(100\ \ ms,\ \ 0)$

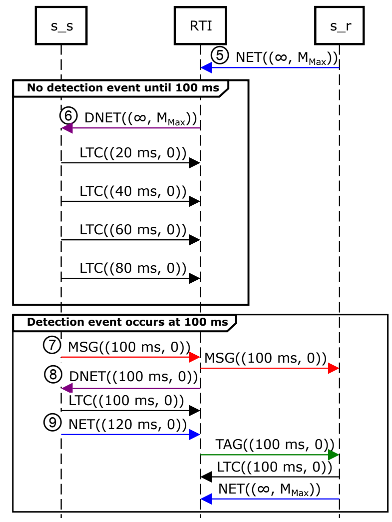

Fig. 4: An execution trace of the program in Fig. 2 with DNET.

Timer Period	**5 ms**	**10 ms**	**20 ms**	**50 ms**	**100 ms**
Baseline	100,161	50,191	25,193	10,195	5,195
Our Solution	677	385	301	288	297

TABLE II: Number of exchanged NET signals during the 500 seconds of runtime with timer periods from 5 ms to 100 ms, using the SparseSender example shown in Fig. 2.

as $\min(N_{s\_r}, H(Q_{s\_r}))$ is $(100\ \ ms,\ \ 0)$. Thus, the RTI sends $\text{DNET}_{s\_s}(100\ \ ms,\ \ 0)$ in ⑧. Upon receiving the new DNET_{s_s}, s_s sends $\text{NET}_{s\_s}((120\ \ ms,\ \ 0))$ in ⑨, as $(120\ ms,\ 0) > (100\ ms,\ 0)$.

VI. Evaluation

We evaluate our approach in comparison with the baseline implementation of LF. We take the example in Fig. 2 as a microbenchmark. We simulate the examples with our approach and the baseline and compare the number of NET signals.

We assume the actual event detection happens every 5 seconds while varying the sensing periods (the timer periods). Note that, in practice, the actual detection usually occurs more sparsely. For example, in a real smart factory or a distance sensing system on a vehicle, a defective product or an object (hopefully) does not appear at intervals of a few seconds.

TABLE II shows the count of the NET signal. Our solution remarkably reduces the number of signals for every example. The number of signals decreases by at most 185 times. We observe that our solution becomes more effective as the sparsity grows (as the timer period decreases while the event detection period is constant).

VII. Conclusion

In this paper, we propose an efficient timing coordination for DDE systems. Our evaluation using an extended version of an open-source DDE system shows the proposed solution significantly reduces the cost of exchanging network signals. We note that the effectiveness of the proposed approach varies depending on the topology and sparsity of the application.

REFERENCES

[1] IEEE, "IEEE standard for modeling and simulation (M&S) high level architecture (HLA)– framework and rules," *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000) - Redline*, pp. 1–38, 2010.

[2] E. Noulard, J.-Y. Rousselot, and P. Siron, "CERTI, an open source RTI, why and how," 2009.

[3] M. Lohstroh, E. A. Lee, S. Edwards, and D. Broman, "Logical time for reactive software," in *Workshop on Timing-Centric Reactive Software (TCRS), in Cyber-Physical Systems and Internet of Things Week (CPSIoT)*. ACM, 2023, Conference Proceedings.

[4] B.-G. Jun, D. Kim, M. Lohstroh, and H. Kim, "Reliable event detection using time-synchronized iot platforms," in *Proceedings of Cyber-Physical Systems and Internet of Things Week 2023*, ser. CPS-IoT Week '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 355–360. [Online]. Available: https://doi.org/10.1145/3576914.3587501

[5] M. Lohstroh, C. Menard, S. Bateni, and E. A. Lee, "Toward a lingua franca for deterministic concurrent systems," *ACM Trans. Embed. Comput. Syst.*, vol. 20, no. 4, May 2021. [Online]. Available: https://doi.org/10.1145/3448128

[6] S. Bateni, M. Lohstroh, H. S. Wong, H. Kim, S. Lin, C. Menard, and E. A. Lee, "Risk and mitigation of nondeterminism in distributed cyber-physical systems," in *ACM-IEEE International Conference on Formal Methods and Models for System Design (MEMOCODE)*, 2023, Conference Proceedings.

[7] S. Jafer, Q. Liu, and G. Wainer, "Synchronization methods in parallel and distributed discrete-event simulation," *Simulation Modelling Practice and Theory*, vol. 30, pp. 54–73, 2013.

[8] K. Rudie, S. Lafortune, and F. Lin, "Minimal communication in a distributed discrete-event system," *IEEE transactions on automatic control*, vol. 48, no. 6, pp. 957–975, 2003.

[9] X. Wang, S. J. Turner, M. Y. H. Low, and B. P. Gan, "Optimistic synchronization in HLA based distributed simulation," in *Proceedings of the eighteenth workshop on Parallel and distributed simulation*, 2004, pp. 123–130.

[10] N. Tampouratzis, I. Papaefstathiou, A. Nikitakis, A. Brokalakis, S. Andrianakis, A. Dollas, M. Marcon, and E. Plebani, "A novel, highly integrated simulator for parallel and distributed systems," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 17, no. 1, pp. 1–28, 2020.

[11] M. Lohstroh, "Reactors: A deterministic model of concurrent computation for reactive systems," Ph.D. dissertation, EECS Department, University of California, Berkeley, Dec 2020. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-235.html

[12] M. Lohstroh, Í. Í. Romeo, A. Goens, P. Derler, J. Castrillon, E. A. Lee, e. R. Sangiovanni-Vincentelli, Alberto", M. Edin Grimheden, and W. Taha, "Reactors: A deterministic model for composable reactive systems," in *Cyber Physical Systems. Model-Based Design*. Cham: Springer International Publishing, 2020, pp. 59–85.

[13] G. A. Agha, *Abstracting Interaction Patterns: A Programming Paradigm for Open Distributed Systems*. Boston, MA: Springer US, 1997, pp. 135–153. [Online]. Available: https://doi.org/10.1007/978-0-387-35082-0_10

[14] C. M. Kirsch and A. Sokolova, *The Logical Execution Time Paradigm*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 103–120. [Online]. Available: https://doi.org/10.1007/978-3-642-24349-3_5

[15] A. Benveniste, P. Caspi, S. A. Edwards, N. Halbwachs, P. Le Guernic, and R. De Simone, "The synchronous languages 12 years later," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 64–83, 2003.

[16] E. A. Lee, J. Liu, L. Muliadi, H. Zheng, and C. Ptolemaeus, "Discrete-event models," *System Design, Modeling, and Simulation using Ptolemy II*, 2014.

[17] P. Donovan, E. Jellum, B. Jun, H. Kim, E. A. Lee, S. Lin, M. Lohstroh, and A. Rengarajan, "Strongly-consistent distributed discrete-event systems," *arXiv preprint arXiv:2405.12117*, 2024.