Special Sessions - Predictable Timing Behavior in Distributed Cyber-Physical Systems

Jian-Jia Chen, Mario Günzel, Dakshina Dasari, Matthias Becker, Edward A Lee, Timothy Bourke

Special Session - Predictable Timing Behavior in Distributed Cyber-Physical Systems

Jian-Jia Chen
TU Dortmund University
Dortmund, Germany
jian-jia.chen@tu-dortmund.de

Mario Günzel
TU Dortmund University
Dortmund, Germany
mario.guenzel@tu-dortmund.de

Dakshina Dasari
Robert Bosch GmbH
Renningen, Germany
Dakshina.Dasari@de.bosch

Matthias Becker
KTH Royal Institute of Technology
Stockholm, Sweden
mabecker@kth.se

Edward A. Lee
UC Berkeley
eal@berkeley.edu

Timothy Bourke
Inria / ENS, PSL University, CNRS
Paris, France
timothy.bourke@inria.fr

Abstract

Ensuring predictable and deterministic behavior in distributed cyber-physical systems (CPS) is essential for guaranteeing safety, reliability, and real-time behavior. However, achieving this predictability is challenging due to network uncertainties, asynchronous execution, and complex timing interactions.

This manuscript is based on a special session at Embedded Systems Week (ESWeek) 2025, which brings together experts to explore in *four presentations* how this uncertainty can be addressed and how to introduce additional determinism into the system to achieve predictable timing behavior in distributed CPS.

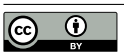
We begin by exploring *cornerstones of timing analysis techniques* to provide end-to-end latency guarantees for distributed systems *(Chen and Günzel)*. Next, we discuss *design strategies for meeting timing constraints*, focusing on how system parameters influence cause-effect chains and how these parameters can be tuned to ensure predictable behavior in industrial automation settings *(Dasari and Becker)*. We then turn to approaches to achieve more predictable system behavior. To that end, we examine *deterministic semantic models* for distributed systems that enable the design of robust and fault-tolerant systems *(Lee)*. Finally, we discuss how *solving constraints for scheduling cause-effect chains* can be used to enforce strict timing guarantees and improve predictability *(Bourke)*.

CCS Concepts

• **Computer systems organization → Embedded and cyber-physical systems**; • **Software and its engineering → Real-time systems software**.

Keywords

End-to-End Latency, Cause-Effect Chain, Distributed Cyber-Physical Systems



EMSOFT '25, Taipei, Taiwan

ACM ISBN 979-8-4007-1993-6/2025/09
https://doi.org/10.1145/3742874.3757086

ACM Reference Format:
Jian-Jia Chen, Mario Günzel, Dakshina Dasari, Matthias Becker, Edward A. Lee, and Timothy Bourke. 2025. Special Session - Predictable Timing Behavior in Distributed Cyber-Physical Systems. In *International Conference on Embedded Software (EMSOFT '25), September 28-October 3, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3742874.3757086

1 Introduction

Modern cyber-physical systems are evolving towards more distributed architectures, driven by the need for scalability and flexibility. Despite its benefits, the shift towards distributed cyber-physical systems (CPS) introduces significant challenges in maintaining predictable and deterministic timing behavior, due to network uncertainties, asynchronous execution, and complex timing interactions.

When functionalities in distributed cyber-physical systems rely on the interaction of multiple distributed tasks, transferring produced/required data between the tasks is required. To formalize these data dependencies, functionalities are usually modeled through cause-effect chains, i.e., a sequence of tasks $E = (\tau_1 \rightarrow \cdots \rightarrow \tau_n)$ where the data of task τ_i is transferred to τ_{i+1} for all $i = 1, \ldots, n-1$. Typical examples are from the area of autonomous driving, for which several tasks such as perception, tracking and prediction have to be executed sequentially [79], based on the output of the previous task.

There are (at least) two execution models for such cause-effect chains. The *time-driven* execution model assumes that tasks are activated independently of each other and use the freshest data, e.g., [7, 8, 13, 31, 33, 36, 39, 42–44, 61, 62, 80, 83, 96]. The *event-driven* model assumes that every datum has to be processed, i.e., the data of the previous task in the chain is passed to the next task by also triggering the activation of an instance of that task [11, 26, 40, 82].

While timing behavior for individual tasks is usually analyzed by *worst-case response time* or *schedulability* guarantees, for cause-effect chains we are interested in the time needed to process data through the complete chain. To that end, different metrics have been proposed and analyzed since 2007 [31], with the most prominent metrics being the Maximum Reaction Time (MRT) and the Maximum Data Age (MDA) [36]. Recently, these metrics has been formally defined using job chains [33], augmented job chains [44] and partitioned job chains [46].

When allowing tasks to communicate unconditionally (so-called *explicit* or *direct* communication), the data can become inconsistent [52]. To mitigate this, different communication semantics with different levels of time predictability have been proposed [52]: *implicit communication*, where data is only propagated when a task instance starts and finishes its execution, and communication under the *Logical Execution Time (LET)* paradigm, where communication occurs at predefined offsets. Over time, multiple analyses have been proposed based on different communication models, using over-approximation to account for the worst-case end-to-end timing behavior, as summarized in the tutorial on end-to-end latency [47].

The importance of predictable timing behavior of cause-effect chains in cyber-physical systems can be found from industrial applications [52], industrial challenges [79], robotic applications [90], and eXtended Reality [97]. This manuscript is based on a special session at Embedded Systems Week (ESWeek) 2025, encompassing four talks in which experts explore strategies for managing timing uncertainty and introducing additional determinism to achieve predictable behavior in distributed CPS.

Summary: Section 2 presents the cornerstones of timing analysis techniques for cause-effect chains to provide end-to-end latency guarantees for distributed systems. Design strategies for meeting timing constraints for cause-effect chains in distributed cyber-physical systems are presented in Section 3, with a focus on how system parameters influence cause-effect chains and how these parameters can be tuned to ensure predictable behavior in industrial automation settings. We then turn to approaches to achieve more predictable system behavior. Section 4 examines deterministic semantic models for distributed systems that ensure consistency and timeliness of distributed CPS. We discuss in Section 5 how to schedule control programs by solving the constraints for scheduling for cause-effect chains can be used to enforce strict timing guarantees and improve predictability. We further offer a short discussion on the challenges in Section 6.

2 Cornerstones of End-to-End Timing Analysis

In distributed CPS, the timing requirement of a sequence of communicating tasks (a so-called cause-effect chain) is the end-to-end latency. After introducing end-to-end latency to the real-time community in 2007, there have been several advancements in the research landscape, including the formal definition of timing requirements in the form of several *end-to-end latency metrics*, the proof of *fundamental properties* describing compositional behavior and relation between different metrics, and *several analytical bounds* on the end-to-end latency. This section outlines these recent cornerstones that help to guarantee the timing behavior of distributed CPS.

End-to-End Latency Metrics. To describe the timing behavior of cause-effect chains, different metrics have been proposed in the literature. Specifically, one of the first specifications of end-to-end latency was provided by Davare et al. [31] in 2007, where they analyzed the data flow from a source node to a sink node. After this initial exploration, Feiertag et al. [36] refine the end-to-end latency semantics with a focus on the cases in which under-sampling and over-sampling occurs. To that end, they introduce the notion of an *output interval* that encompasses the time where output based on the same data source can occur, and of an *input interval* where any data captured in this interval will be used by the system to create an output during the output interval (cf. [36, Figure 7]). Based on these input and output intervals, Feiertag et al. [36] defined different metrics for the end-to-end latency, namely First-to-First, First-to-Last, Last-to-First, and Last-to-Last. Among these, two key metrics are emphasized: *First-to-First* latency, also referred to as the **Maximum Reaction Time (MRT)**, is the maximum time between the beginning of the input interval and the beginning of the output interval for any data source. *Last-to-Last* latency, also referred to as the **Maximum Data Age (MDA)**, is the maximum time between the end of the input interval and the end of the output interval for any data source. However, there is an imbalance in the definition of the input and output intervals by Feiertag et al. [36]. Specifically, while the input interval is formulated as a *passive* arrival of data to the system, the output interval is formulated as an *active* generation of data by the system. This imbalance is resolved in [44, 45, 76] by enlarging the output interval. To distinguish the metrics based on the different definitions of input and output intervals, Günzel et al. [47] propose a refinement into the Maximum Reaction Time (MRT), Maximum Reduced Reaction Time (MRRT), Maximum Data Age (MDA), and Maximum Reduced Data Age (MRDA), as summarized in [47, Figure 7].

While the different metrics are initially formalized using the notion of *timed paths* [36], an alternative formalization of the age latency is developed by Martinez et al. [75]. Specifically, they determine *publishing points* and *reading points* to derive *basic paths*, which are later also referred to as *primary job chains* [74]. However, to the best of our knowledge, this modelling of timing behavior is limited to systems using the LET communication paradigm, i.e., requiring a high level of time predictability. Dürr et al. [33] provide an alternative definition of the end-to-end latency in terms of *job chains*, applicable to implicit communication. To that end, they distinguish between immediate forward job chains, to track the progression of data, and immediate backward job chains, to track the origin of data. The notion of job chains has later been extended to *augmented job chains* to cover the full time interval including the input and output of data [44] and to allow all kinds of communication semantics. Recently, Günzel et al. [46] define *partitioned job chains* as a more efficient generalization of augmented job chains to model the different end-to-end latency metrics.

Besides the typical end-to-end latency metrics, where the data propagation along individual cause-effect chains are analyzed, the time-driven execution model provides a more holistic view on the propagation of data packets [2], with an important metric being the flow completion time [32, 53]. Furthermore, the *worst-case time disparity*, where the maximum difference among timestamps from different sensors is analyzed, has gained some interest [56, 93]. When data propagation becomes less reliable, e.g., due to response time randomness or failure probabilities, then probabilistic guarantees on the end-to-end latency are of interest [49, 50, 69, 85].

Analytical Bounds. While the definition of metrics is applicable to both time-driven and event-driven execution models, their analytical approaches are fundamentally different.

Regarding time-driven execution models, we distinguish for periodic and sporadic systems. For periodic systems, task instances are released with a fixed period. Hence, they inherently provide a large

degree of predictability. Therefore, analytical approaches exploit that predictability by unrolling a safe analysis window (commonly one or two hyperperiods) and quantify the length of job chains starting in that analysis window. Prime examples for such analyses are the results from Becker et al. [8, 9] for MRDA and Kloda et al. [62] for MRT. Recent approaches focus on more dedicated solutions [42, 45, 45, 63, 78] and on speeding-up of the computation process [14, 46]. On the other hand, sporadic schedules are not repetitive, and, hence, unrolling the schedule becomes infeasible and the uncertainty in the timing behavior has to be accounted for using analytical over-approximation. A prime example is the analytical bound on the end-to-end latency provided by Davare et al. [31] under implicit communication. The bound was further tightened and extended to the MRDA by Dürr et al. [33]. For tasks under LET communication, Hamann et al. [52] provided an upper bound for the MRT. We note that although the analyses by Davare et al. [31] and Hamann et al. [52] are originally formulated for periodic tasks, their bounds are applicable to sporadic tasks as well.

Fundamental Properties. While analytical bounds are usually derived for a specific system model with a given communication policy and release behavior, as discussed above, recently two fundamental properties have been developed, which are universally applicable. The first is a compositional property that allows cutting a cause-effect chain into smaller segments to be analyzed individually, and the second is the equivalence between MRT and MDA. The mentioned properties are fundamental because they hold for any communication and scheduling mechanism that adheres to the two requirements stated [47, Section 3]. Hence, it covers time-driven like periodic or sporadic tasks under implicit communication or LET, as well as event-driven systems like systems using the ROS 2 mechanism.

The *compositional property* [45] states that the end-to-end latency of any cause-effect chain $E = (\tau_1 \rightarrow \cdots \rightarrow \tau_n)$ can be composed of its subchains, i.e.,

$$MRT(E) \leq MRT(E_1) + MRT(E_2) \tag{1}$$

$$MDA(E) \leq MDA(E_1) + MDA(E_2) \tag{2}$$

with $E_1 = (\tau_1 \rightarrow \cdots \rightarrow \tau_k)$ and $E_2 = (\tau_{k+1} \rightarrow \cdots \rightarrow \tau_n)$. Please note that while the original work [45] uses a modelling of the end-to-end latency using *valid* job chains, the property is moved to the nowadays more common modelling using *warm-up* in [47]. The compositional property can be used to derive safe upper bounds in case segments of the cause-effect chain require different analytical treatments [48]. Furthermore, it allows to exploit the time-predictability of the segments while over-approximating the communication behavior between the segments, e.g., if the cause-effect chain is distributed over multiple asynchronized components [45].

While MRT and MDA were mostly analyzed independently, as they were believed to be inherently different, starting from 2019, the analytical relation between MRT and MDA has been further explored. More specifically, Dürr et al. [33] showed that an analytical bound for MRT in sporadic systems also holds for the MRDA, and Günzel et al. [44, 45] showed that the MRT is an upper bound for the MDA under a more general definition. While the literature provides empirical observations that MRT and MDA coincide in specific settings, e.g., in the absence of over- and undersampling [3, Section 3.6.2, p. 114] or in ROS 2 [90], the *equivalence* of MRT and MDA[1] is formalized by Günzel et al. [46] in 2023:

$$MRT(E) = MDA(E) \tag{3}$$

The equivalence eases the specification and verification of timing constraints in industrial systems, and can lead to more efficient analyses, as demonstrated in [46]. Furthermore, it contributes to more predictable timing behavior by establishing the formal relationship between MRT and MDA.

3 Design Strategies to Meet End-to-End Timing Requirements of Cause-Effect Chains

The end-to-end latency of cause-effect chains in applications is influenced by a range of parameters—some defined during system design, others adjusted during system integration to meet specific latency constraints. These applications typically comprise numerous software functions, often developed by separate teams, and built upon domain-specific middleware layers that depend on the underlying operating system and communication stacks. For instance, robotic or autonomous driving systems commonly use ROS2, which runs on a POSIX-compliant operating system, typically Linux, and relies on DDS (Data Distribution Service) for inter-node communication over TCP/IP or UDP protocols. Once all components are integrated, the resulting system consists of multiple, often overlapping, cause-effect chains, where a single task may contribute to several chains simultaneously. Ensuring that each of these chains satisfies its individual end-to-end latency requirements is a complex challenge—especially given the interplay between application logic, middleware behavior, and system-level resource constraints.

While offering crucial flexibility and modularity for modern automotive Electrical/Electronic (E/E) architectures like those in Mercedes-Benz MBUX or AUTOSAR Adaptive Platform, middleware solutions such as SOME/IP, DDS (e.g., RTI Connext, Apex.AI's Apex.Grace/ROS 2), and gRPC introduce significant variability into the system's temporal behavior, particularly for end-to-end latency across critical cause-effect chains. This unpredictability stems from several key factors. Message queuing and buffering delays are prevalent, especially in DDS-based systems, where QoS settings like reliability and history depth directly influence delivery timeliness. Non-deterministic scheduling and dispatching further compound this, as seen in ROS2's default executor callback patterns or the dynamic scheduling policies of Adaptive AUTOSAR, which can delay processing of critical sensor data. Moreover, inherent network protocol overheads (e.g., SOME/IP's headers, DDS's RTPS reliability, gRPC's HTTP/2 framing) introduce variable latency and jitter. Finally, dynamic service discovery (like SOME/IP-SD or DDS Discovery Protocol) and dynamic resource binding (as in DDS or ZeroMQ for runtime communication partner changes) add initial and runtime unpredictable delays, making it challenging to guarantee the worst-case communication times essential for safety-critical automotive functions. To fully guarantee timeliness of cause-effect chains, the middleware layers either need to handle all possible scenarios or middleware and/or application parameters need to be configured in such a way that end-to-end latencies are guaranteed.

[1]The equivalence result can be transferred to MRRT and the MRDA as well [46, 47].

Cross-layer Approach for Predictable Execution: Predictability presents a complex, multi-layered challenge that demands coordinated interventions across the software stack. The strategies discussed here reflect a systematic effort spanning operating systems, runtime environments, communication protocols, and high-level frameworks to achieve deterministic behavior. However, as illustrated earlier, numerous open issues remain and many a time point-solutions are presented, without considerations of the underlying runtime stacks. Addressing these requires a more holistic, system-level perspective that accounts for sources of non-determinism across layers representative of practical use-cases.

ROS2 Middleware: The ROS2 executor (callback dispatcher) is known to have multiple issues contributing to non-determinism as shown by Casini et.al [22]. Furthermore with [59], the authors demonstrate that the issue of priority inversion and unpredictable delays arise from the lack of consistent priority propagation across the ROS2 stack architecture all the way down to the kernel layer. To deal with this, they propose CROS-RT, a cross-layer priority scheduling framework that enforces consistent priority-driven scheduling, mitigating priority inversion and ensuring predictable end-to-end latency for high-priority tasks. Choi et al. [27] introduced a priority-driven, chain-aware scheduler for ROS2, directly linking ROS2 callback prioritization to the timing requirements of task chains.

Communication Middleware: Data Distribution Service (DDS) is a key middleware for modern interconnected applications (IoT, Edge, Cloud) and also in automotive and robotics middleware (e.g., ROS2). With [84], the authors provide an abstract, generalizable model for DDS systems and provide a real-time analysis that allows bounding the data-delivery latency of DDS messages.

Operating System (OS) / Scheduling Layer. The QNX operating system is the de facto standard operating system in the automotive domain for safety-critical high-performance platforms and infotainment, and is certified to the highest safety assurance level (ASIL-D). In addition to preemptive fixed-priority scheduler, QNX also implements a resource reservation algorithm that guarantees processor bandwidth for a group of tasks, called Adaptive Partitioning Scheduler (APS) to address the needs of mixed-criticality tasks on a centralized vehicle computer in the automotive domain. Dasari et al. pointed out the shortcomings in the existing APS scheduler [30] and provide a precise description of the APS mechanism [29], validated by experiments on a real platform. A response time analysis is proposed that allows to bound the end-to-end latency of event-driven cause-effect chains that span multiple APS-partitions and cores.

Predictable Inter-Process Communication (IPC):. Beyond execution, OS-level communication mechanisms are critical. QNET, QNX's native networking manager, coupled with Synchronous Message Passing (SyncMP), provides fundamental IPC. Becker et al. [6] extended RTA to local and inter-node SyncMP communications, incorporating task mapping within APS partitions, thereby establishing predictable communication latencies.

Commercial Runtime Environments. Commercial offerings like ETAS Deterministic Middleware Solutions (DMS) [1] provide integrated solutions for deterministic execution and supports both time-triggered and event triggered execution. ETAS DMS guarantees data consistency by treating activity (each activity is a group of runnables) inputs as immutable once started, and output data as batched and released only upon completion. This "all or nothing" approach to data visibility between activities, combined with serialized execution, prevents parallel conflicts and ensures data integrity at activity borders and helps with deterministic recompute irrespective of variations in data arrivals.

Similarly, TTTech Auto's MotionWise safety middleware [92] is a safety-certified, deterministic software platform that serves as an abstraction layer between application software and the underlying hardware and system software. MotionWise supports both event-driven scheduling and time-triggered scheduling and uses time-synchronized communication protocols like Time-Sensitive Networking (TSN) allowing the enforcement of precise communication and execution schedules across distributed systems.

Elektrobit EB Corbos [34, 35] supports implementation of LET-based system behavior (aka System LET) across distributed ECUs enabling end-to-end latencies in cause-effect chains spanning different nodes.

Runtime execution layer. Various techniques at the runtime execution layer can also modify task parameters to ensure timely completion of data flows. Job-Level Dependencies (JLD) as proposed by Becker et al. [8, 11] specify precedence constraints between selected task instances, guaranteeing end-to-end latency. While this approach is agnostic of the scheduling algorithm, it requires runtime mechanisms [60, 61] or JLDs to be resolved at design time by merging multiple tasks statically together [5].

When LET is used for communication, job-level determinism is guaranteed, i.e. during execution, always the same jobs communicate with each other. Offset/Phase Assignment Heuristics configure task parameters to optimize for minimal latency in task chains, finding the best alignment for task execution. Martinez et. al. [75] proposed a heuristic for phase assignment by evaluating all non-equivalent offset assignments of tasks in a chain, while Günzel and Becker [43] developed an optimal phase assignment for semi-harmonic task chains. Furthermore, jitter reduction techniques address variations in end-to-end latency, such as adding copy-tasks to multi-rate logically executed time (LET) chains to mitigate jitter, as shown by Bini et al. [16]. Optimization for implicit communication chains, less explored, includes determining period values for new tasks within legacy systems to maintain overall latency constraints [10].

Generalized LET Variants. Traditional LET models schedule read or write operations at period boundaries. More generalized models allow these operations to occur earlier or later, enabling finer-grained LET interval configuration. Paladino et al. [77] configured LET parameters based on task response times, optimizing chain latency by selectively adjusting task priorities. Maia et al. [74] integrated scheduler information into later design phases to shorten and shift LET intervals for latency reduction. Wang et al. [94] proposed a comprehensive framework for optimizing LET intervals across multiple metrics, including end-to-end latency.

Industrial Automation Systems. Industrial automation applications are inherently distributed consisting of chains spanning sensing, control and actuation tasks; however in traditional systems the core control function is executed as a monolithic unit on a proprietary industrial computer directly connected to the I/O. Traditional execution runtimes in commercial platforms (Beckhoff TwinCAT, Siemens TIA Portal with S7, Bosch Rexroth ctrlX CORE [18], and Codesys-based systems) ensure determinism through a combination of cyclic and fixed priority task scheduling combined with watchdogs to detect timing violations, real-time fieldbus protocol like EtherCAT (used by TwinCAT, ctrlX, and Codesys) or PROFINET (used by Siemens), and hardware-assisted time synchronization. These proprietary stacks minimize runtime variability by avoiding or tightly constraining dynamic behaviors such as service discovery, message queuing, or asynchronous callbacks. This architectural discipline enables industrial automation systems to guarantee end-to-end timing—even in complex, multi-node control environments—providing reliable and predictable control performance.

However, the increasing move toward virtualizing control functions and shifting processing to edge or cloud platforms adds complexity to the traditionally well-controlled industrial automation environment, resulting in greater timing unpredictability. To address this, control architectures are often designed as hierarchical or hybrid systems: time-critical, low-level control remains on dedicated real-time hardware located near the process (such as PLCs or edge devices), while higher-level, less time-sensitive tasks are executed on cloud or general-purpose hardware. This separation ensures that critical control loops maintain determinism while still leveraging the scalability offered by cloud computing. Despite using standard Ethernet/IP networks, these systems leverage TSN features like time-aware scheduling and traffic shaping to ensure low jitter and bounded latency. Protocols such as OPC UA PubSub build on TSN to achieve sub-millisecond end-to-end precision, enabling coordinated, time-synchronized control across distributed automation nodes. Additionally, emerging containerization technologies such as WebAssembly are being explored within industrial automation for their advantages in rapid startup times, lightweight design, and portability across diverse platforms [81]. Furthermore, distributed applications must be explicitly designed to tolerate timing anomalies such as out-of-order, dropped, or delayed message deliveries across various layers of the stack—including the kernel, middleware, and application interface—by adopting deterministic timing paradigms that ensure functional correctness and timing predictability even under adverse conditions.

Symbiotic Relationship of Applications and Infrastructure for Distributed Control Resilience. To achieve resilience, distributed control systems must enable dynamic adaptation. Applications should leverage infrastructure feedback on network conditions (e.g., degradation) to implement multimodal operation, planned degradation, and failover. This is critical for split-control architectures, where local controllers can manage message loss to maintain desired latency. Conversely, the infrastructure must proactively allocate resources based on application state (health, KPIs) to meet end-to-end latency requirements and optimize overall QoS for critical applications.

4 Consistency and Timeliness in Distributed CPS

As a result of the intrinsic uncertainties of distributed systems, many designers throw up their hands and decide that they do not need deterministic models, opting instead for easy-to-use publish-and-subscribe buses, actor networks, and remote procedure calls. But these choices come with considerable costs. Because the underlying semantic models are nondeterministic, systems do not define a single correct response to a particular input stimulus. This makes it much more difficult to form consistent views of system state across a distributed system, to detect faults at runtime, and to design regression tests that protect against errors at design time. Is an observed behavior one of the many allowed by the semantic model? Or is it the result of a fault in the system? Moreover, anomalies in such nondeterministic semantic models are often rare and difficult to reproduce, which means that systems can perform well with extensive testing and then fail in the field.

A key challenge in distributed systems is to form a consistent view of the state of the system across distributed components. There are many applications that require such consistency. For example, autonomous vehicles should not enter an intersection without a consistent view across vehicles of the state of the intersection.

Loosely, consistency is agreement on shared information, where the information can change over time. In certain cases, if the information changes can be modeled by a monotonic function in some partial order, then *eventual* consistency can be achieved without coordination [64]. However, "eventual" may not be good enough. If timing matters, then consistency requires coordination [70]. Moreover, fundamental limits show that, as network latencies increase, the time taken to form a consistent view increases [66].

Most frameworks for coordinating concurrent and distributed software components provide little or no help towards ensuring consistency or managing the timing costs. Publish-and-subscribe frameworks (such as ROS 2 and MQTT), actor frameworks (such as in Erlang, Akka, Orleans, and CAF), and remote procedure call frameworks (such gRPC and Apache Thrift) are particularly weak in this regard. There are better alternatives.

Given now that coordination is required, what coordination mechanism should we use? One strategy is to centralize shared information, but this has many drawbacks. For one, it creates a single point of failure, where failure of the central store can doom a system. Moreover, it requires very careful design, often well beyond the skill set of system engineers. Given that it will take time to access central information, for example, how can a component be sure the information is still valid when it receives it?

We propose that a basic and potentially ubiquitous coordination mechanism, clock synchronization [57], can provide a foundation for distributed systems with consistency and timing guarantees, up to the fundamental limits. Clock synchronization with bounded error provides a foundation for time-stamping messages and providing a useful global semantic ordering that can guarantee consistency.

The Lingua Franca (LF) coordination language [73] makes timestamps a central feature, providing a logical timeline [72] that enables deterministic concurrency. Concurrent components exchange

timestamped messages, and every component processes events in timestamp order.

In principle, the deterministic concurrency of LF can be achieved without coordination if timing does not matter and interaction is not needed [70]. But for CPS, timing and interaction are always needed, so this is not useful. Some level of coordination becomes essential.

When LF programs are distributed across networks, two distinct coordination mechanisms are provided in the open-source code base [4]. A *centralized coordinator* mediates all interactions in order to ensure deterministic semantics without relying on clock synchronization or bounded communication latencies. A *decentralized coordinator*, in contrast, relies on clock synchronization and enables tradeoffs between ensured consistency and timeliness [65].

The decentralized coordinator in LF is much more interesting for CPS applications. It has no single point of failure, no centralized bottleneck, and can provide a full range of consistency strategies from fully guaranteed (at the fundamentally unavoidable price of potentially unbounded latency) to guaranteed under the assumption of bounded network latencies. With the weaker guarantees, the LF mechanisms also provide hooks for application-specific fault handlers to be invoked when assumptions about network latency bounds are violated or when components fail.

In summary, the Lingua Franca coordination language, with its relatively new decentralized coordinator, can realize sophisticated distributed computing patterns that enable designers to manage the fundamental tradeoffs between consistency and timeliness.

5 Scheduling Rate-Synchronous Control Programs

In some distributed CPS, individual tasks are specified using languages like Lustre/Scade [28, 51] or MathWorks Simulink [91]. Each task is then described as a functional schema from inputs to outputs, sometimes called a block diagram. When tasks are executed cyclically and atomically, it is natural to model them as functions from streams of inputs, coming from sensors or other tasks, to streams of outputs, going to other tasks or actuators. A stream function is intrinsically atomic and deterministic, but maintaining these properties for a set of tasks with diverse activation rates and execution times is challenging. The LET model, discussed in Section 2, does it by specifying the delay between the input consumption and output production of a task. As described in Section 4, the LF language does it by tagging stream elements with timestamps and employing coordinators. Individual tasks can also be composed into a single Lustre program that is implemented with preemptive scheduling either by preserving the source semantics using a communications protocol [25, 87] or by incorporating real-time properties into the source language [37, 38]. Real-time scheduling permits less frequent tasks to execute for longer but, as described earlier, complicates reasoning about semantics preservation and end-to-end latency.

Flight control software developed at Airbus comprises thousands of individual tasks, each programmed with Lustre/Scade, communicating via hundreds of thousands of named signals. The cause-effect chains that are critical for control system performance are identified and each is associated with a maximum end-to-end latency bound. A static schedule is produced by solving an Integer Linear Programming problem and encoded as a sequential program that executes every 5 ms. Individual tasks may be activated less frequently, but they must still execute completely within a single cycle at the base period. In terms of timing behavior, an implementation is correct if (i) the sum of task execution times in any cycle is less than the base period and (ii) the end-to-end latency bounds are respected. This approach is systematized in recent variations of Lustre [19, 54] that add features for expressing the overall program as a composition of individual tasks. While such top-level schemas are normally too large and complex to visualize as block diagrams, treating them as programs allows to formalize static constraints as typing rules, required dynamic behavior as a semantic model, and code generation as a compilation problem.

The Lustre variation that we call the *rate-synchronous model* [19] formalizes the Airbus approach. The execution rates of individual tasks are expressed as unit fractions $1/n$ of the base rate. For instance, when the base period is 5 ms, a task at rate $1/4$ executes every 20 ms. A fast-to-slow rate transition specifies how to filter an incoming stream by taking, for example, the second of every four incoming values. Conversely, a slow-to-fast rate transition repeats every incoming value a fixed number of times after producing a stated number of initial values. For communications at the same rate, there is an operator for reading the previous value of a named stream. These three operations and the ability to instantiate externally defined functions suffice to represent a set of communicating tasks. A simple set of rules [19, Figure 2(b)] formalizes the required relations between unit fraction clocks. The semantic rules are equally simple [19, Figure 2(a)]. They associate every variable and expression with an infinite sequence and, ultimately, a whole program with a function from a list of input sequences to a list of output sequences.

A rate-synchronous program is compiled into a sequential program in two steps: *scheduling* assigns a phase to each task and *microscheduling* orders the execution of tasks within a single cycle. The resulting program maintains a counter modulo the hyperperiod *hp* (the least common denominator of its rates) and uses it to determine which task step functions to call in any cycle. Step functions are called one-by-one to read and write the variables that implement the signals from the source program, and to update any internal state. Since only a single scalar variable is allocated for each signal, the system is essentially a Kahn process network [58] with one-place buffers: executing the tasks in any order that respects their communication dependencies will calculate the stream function defined by the source semantics.[2]

For a task at rate $1/n$, scheduling can choose any phase $0 \leq p < n$ that respects the communication dependencies. For example, if this task reads the second of every four values of another task at rate $1/m$ with phase $0 \leq q < m$, it must be scheduled after the phase $m + q$ and before the phase $2m + q$. For certain phase choices, the two tasks are executed in the same cycle, and microscheduling then determines which runs first. Scheduling and microscheduling must thus somehow agree on the "instantaneous relative ordering", or *concomitance*, for every pair of communicating tasks. For

[2]Not all programs can be implemented in this way; some require rewriting to induce additional buffering [19, §2.2].

the example: forward concomitance means that microscheduling must place the writer before the reader and the scheduling constraint is $m + q \leq p < 2m + q$; backward concomitance means that microscheduling must place the reader before the writer and the scheduling constraint is $m+q < p \leq 2m+q$. Typically, microscheduling orders tasks "fast first", that is, from fastest to slowest, and thus fast-to-slow rate transitions have forward concomitance, slow-to-fast rate transitions have backward concomitance, and same-rate communications default to forward concomitance when the current value is required and backward concomitance when the previous value is required [21].

Given the same inputs, all correct schedules calculate the same output sequences, but their fine-grained timing properties may differ. For two tasks at the same rate $1/n$, the difference between their actual start times, or end times, can be up to n times the base period. Unsurprisingly, the precision of sensor samples and actuator updates depends on the chosen rate. Also, the worst-case execution time of the generated code is the maximum of the sums of task execution times in any cycle, which must be less than the base period for the implementation to be correct. Lastly, for some system functions, designers may initially be less concerned about whether, for instance, tasks sample the second or third values of an incoming stream, or even the current or last values [21, 54, 95], and more concerned about respecting end-to-end latency bounds. In this case, communication dependencies are sometimes relaxed during scheduling, though the result is still a deterministic program.

For worst-case execution time, and other platform constraints, the source language provides features for declaring named resources, stating the quantities required by external tasks, and requesting that resource sums be bounded per cycle or balanced across all cycles. This mechanism is simple and readily translated into ILP constraints. In practice, implementers analyze the worst-case execution time of the code generated for individual tasks to derive a weight value that is declared in the main program. Scheduling chooses phases that respect the communication dependencies and minimize the total weight per cycle. The worst-case execution time of the generated schedule is then reevaluated and compared to the base period. The fact that the estimated and scheduled worst-case execution times differ may waste processor time, but does not risk correctness.

For end-to-end latencies, designers declare cause-effect chains by listing tasks together with an upper bound. The list must describe a path through the underlying dependency graph. The upper bound is an integer that represents the number of cycles, at the base period, from an activation of the first task in the path, through related activations of the ensuing tasks. The ILP encoding is not simply the sum of minimum pairwise latencies. Rather it must consider paths through task activations over an entire hypercycle, and possibly continued across hypercycles for tasks that read the previous value of a signal [19, §3.4]. For every task w in a chain, we introduce an *instance variable* $0 \leq i_w < {}^{hp}/_{\text{period}(w)}$ to represent which of its activations participates in an instance of the chain. For each pair of consecutive tasks w and r, we add an equation:

$$i_w \cdot \text{period}(w) + p_w + lat_{w,r} = i_r \cdot \text{period}(r) + p_r,$$

which relates the instants of writing and reading, each given as a multiple of period plus phase, with the latency between them. Separate bounds on the *lat* variables limit their "elasticity", and each instance variable, except the first and last ones, is constrained by two equations, thereby modeling a path of value propagation. Finally, the end-to-end latency is simply the sum of *lat* variables.

The variables and constraints for cause-effect chains are added to those for communication dependencies and resources to generate an ILP problem. A solution to this problem gives a phase assignment that implements the stream semantics, distributes calculations across cycles, and respects end-to-end latency bounds. The compiler validates that a given phase assignment respects these requirements and generates a sequential program in the form described above using an adaptation of the standard clock-directed compilation scheme [15].

6 Conclusions and Challenges for Predictable Timing Behavior

We have presented several means of analyzing, designing, and optimizing distributed cyber-physical systems regarding their timing behavior, especially when considering cause-effect chains. While this manuscript focuses on chains from the timing perspective, i.e., the end-to-end latency of cause-effect chains, works such as [41] provide a different angle by considering the logical perspective, i.e., to identify causal chains for system failures. How these two different perspectives can be combined remains an open problem.

Furthermore, the design of applications subject to end-to-end latencies faces several challenges that arise from the growing complexity of software stacks and hardware platforms. Today's industrial systems rely on heterogeneous software stacks (e.g. ROS2, DDS, etc.) and heterogeneous hardware platforms. At the same time, legacy systems must be integrated and often constitute a large part of the system.

Predictability of Analysis. The nature of predictability for event-driven systems is much different from the predictability present in time-driven systems. That is, while the arrival pattern (i.e., the time when task instances arrive in the system) is less predictable, the communication behavior (i.e., which jobs communicate with each other) is more predictable. Therefore, the main challenge is to exploit the additional predictability in the communication behavior while bounding the uncertainty in the arrival pattern. An ad-hoc approach is to apply the analyses for sporadic systems in case the inter-arrival time of tasks can be bounded. However, this introduces inherent pessimism because it disregards the increased predictability from the communication behavior. Therefore, more dedicated analyses have been provided [11, 26, 40, 82]. On the edge of event-driven and time-driven systems are those using the ROS 2 middleware, because it allows the specification of both time-triggered and event-triggered nodes. Dedicated analyses for ROS 2 have been proposed in [17, 23, 55, 86, 88–90].

Design Strategies. A fundamental challenge is the design of applications that consist of multiple interconnected cause-effect chains. The main focus of the literature, so far, is on the optimization and design of single cause-effect chains [8, 16, 43, 75]. The optimization of interconnected cause-effect chains is especially challenging if different application modes are considered that affect a different subset of chains at a time.

Modern high-performance platforms offer heterogeneous compute resources. A cause-effect chain can be distributed across several types of compute resources with different scheduling or arbitration mechanisms. The analysis and optimisation of cause-effect chains on such platforms resembles distributed systems, and similar design strategies might be effective. An additional challenge on such platforms is that it is generally not possible anymore to determine worst-case execution times due to the platform complexity requiring runtime monitoring support [24]. While the configuration of the application is initially decided at design time, it might be changed at runtime based on observed runtime statistics. The complexity of monitoring, runtime decision making, reconfiguration, and their effect on the end-to-end latency offer interesting research directions.

Consistency and Predictability. For distributed systems, where components communicate over networks, it is no longer sufficient to bound execution times of software. It also becomes necessary to consider communication latencies and the possibility of lost communication (equivalent to very large latencies). Time-sensitive networking (TSN) [71] may become an attractive option, particularly since it provides an underlying clock synchronization fabric and the possibility for bounded communication latency. It can be quite complicated to configure, however, and is primarily designed for wired networks, not wireless.

Rate-Synchronous Model. The rate-synchronous model, described in Section 5, requires designers to divide the source design into short tasks. An alternative would be to selectively inline tasks to increase the granularity exposed to the scheduling algorithm. This, however, increases the size of the generated ILP problem and the memory needed for global signals, and complicates traceability and memory access patterns. Is there a better way to overcome the incompatibility between a hierarchical source program and allocation of global resources needed to implement it?

The model and compilation scheme target a single computer. Could it be usefully extended to treat distributed systems? The required synchronizations and the cost of constraint solving may be too limiting for large-scale distributed CPS. But perhaps rate-synchronous nodes or sets of tasks could be integrated as reactors within a larger system designed using the principles or language described in Section 4? This may also provide a way to decompose constraint solving into smaller problems: the challenge would be to incorporate system-wide end-to-end latency constraints while solving individual scheduling problems. Can this be done iteratively and monotonically?

As mentioned in Section 5, designers may sometimes prefer to state required end-to-end latencies and use constraint solving to decide how to communicate intermediate values. Ideally, the modeling language would help to ensure that this does not compromise the essential determinism of the source program. Rate-synchronous programs, however, only provide limited information on the underlying control problem, namely, the type of signals (boolean, integer, or floating-point) and the activation rates. Stronger guarantees could potentially be offered by starting from a hybrid programming language, like MathWorks Simulink [91], continuous actor-oriented models [67, 68], or synchronous languages with resettable ordinary differential equations [12, 20]. Is there a systematic way to move from such specifications to rate-synchronous programs with appropriate end-to-end latency constraints?

Acknowledgments

This work is part of a project (PropRT) that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 865170). This work was partially funded by the German Federal Ministry of Research, Technology and Space of Germany (BMFTR) in the course of the 6GEM research hub under grant number 16KISK038. This work was partially funded by the Swedish Research Council (VR) under the project nr. 2023-04773 and by Sweden's Innovation Agency via the NFFP8 project 2024-01267: PARTI. Additional funding was also provided by the US National Science Foundation, #CNS-2233769 (Consistency vs. Availability in Cyber-Physical Systems), and the iCyPhy Research Center (Industrial Cyber-Physical Systems) at UC Berkeley.

References

[1] Etas deterministic middleware solution. https://edms.etas.com/, 2025. Accessed: 2025-08-06.

[2] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. Fernández-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proc. VLDB Endow.*, 8(12):1792–1803, 2015.

[3] AUTOSAR. Specification of timing extensions (AUTOSAR CP R22-11). https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_TPS_TimingExtensions.pdf, 2022.

[4] 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.

[5] M. Becker. Meeting job-level dependencies by task merging. In *29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 792–798, 2024.

[6] M. Becker, D. Dasari, and D. Casini. On the QNX IPC: Assessing predictability for local and distributed real-time systems. In *29th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 289–302, 2023.

[7] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte. Mechaniser-a timing analysis and synthesis tool for multi-rate effect chains with job-level dependencies. In *7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems WATERS*, volume 16, 2016.

[8] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte. Synthesizing job-level dependencies for automotive multi-rate effect chains. In *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 159–169, 2016.

[9] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture - Embedded Systems Design*, 80:104–113, 2017.

[10] M. Becker and S. Mubeen. Timing analysis driven design-space exploration of cause-effect chains in automotive systems. In *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, pages 4090–4095, 2018.

[11] M. Becker, S. Mubeen, D. Dasari, M. Behnam, and T. Nolte. A generic framework facilitating early analysis of data propagation delays in multi-rate systems. In *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–11, 2017.

[12] A. Benveniste, T. Bourke, B. Caillaud, and M. Pouzet. Divide and recycle: Types and compilation for a hybrid synchronous language. In J. Vitek and B. De Sutter, editors, *Proc. 12th ACM SIGPLAN Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES 2011)*, pages 61–70, Chicago, USA, Apr. 2011. ACM Press.

[13] A. Benveniste, P. Caspi, P. L. Guernic, H. Marchand, J.-P. Talpin, and S. Tripakis. A protocol for loosely time-triggered architectures. In *EMSOFT*, pages 252–265, 2002.

[14] R. Bi, X. Liu, J. Ren, P. Wang, H. Lv, and G. Tan. Efficient maximum data age analysis for cause-effect chains in automotive systems. In *DAC*, pages 1243–1248. ACM, 2022.

[15] D. Biernacki, J.-L. Colaço, G. Hamon, and M. Pouzet. Clock-directed modular code generation for synchronous data-flow languages. In *Proc. 9th ACM SIGPLAN*

Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES 2008), pages 121–130, Tucson, AZ, USA, June 2008. ACM Press.

[16] E. Bini, P. Pazzaglia, and M. Maggio. Zero-jitter chains of periodic LET tasks via algebraic rings. *IEEE Transactions on Computers*, 72(11):3057–3071, 2023.

[17] T. Blaß, D. Casini, S. Bozhko, and B. B. Brandenburg. A ROS 2 response-time analysis exploiting starvation freedom and execution-time variance. In *RTSS*, pages 41–53. IEEE, 2021.

[18] Bosch Rexroth. *ctrlX Automation by Rexroth*, 2023.

[19] T. Bourke, V. Bregeon, and M. Pouzet. Scheduling and compiling rate-synchronous programs with end-to-end latency constraints. In A. V. Papadopoulos, editor, *35th Euromicro Conf. on Real-Time Systems (ECRTS 2023)*, volume 262 of *Leibniz Int. Proc. in Informatics (LIPIcs)*, pages 1:1–22, Dagstuhl, Germany, July 2023. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[20] T. Bourke and M. Pouzet. Zélus: A synchronous language with ODEs. In C. Belta and F. Ivancic, editors, *Proc. 16th Int. Conf. on Hybrid Systems: Computation and Control (HSCC 2013)*, pages 113–118, Philadelphia, USA, Apr. 2013. ACM Press.

[21] T. Bourke and M. Pouzet. Lustre, fast first and fresh. *IEEE Embedded Systems Letters*, 17(2):119–122, Nov. 2024. Articles from the workshop on Time-Centric Reactive Software (TCRS 2024).

[22] D. Casini, T. Blaß, I. Lütkebohle, and B. B. Brandenburg. Response-Time Analysis of ROS 2 Processing Chains Under Reservation-Based Scheduling. In S. Quinton, editor, *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:23, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[23] D. Casini, T. Blaß, I. Lütkebohle, and B. B. Brandenburg. Response-time analysis of ROS 2 processing chains under reservation-based scheduling. In *ECRTS*, volume 133 of *LIPIcs*, pages 6:1–6:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[24] D. Casini, P. Pazzaglia, and M. Becker. Managing real-time constraints through monitoring and analysis-driven edge orchestration. *Journal of Systems Architecture*, 163:103403, 2025.

[25] P. Caspi, N. Scaife, C. Sofronis, and S. Tripakis. Semantics-preserving multi-task implementation of synchronous programs. *ACM Trans. Embedded Computing Systems*, 7(2):15:1–40, Jan. 2008.

[26] H. Choi, M. Karimi, and H. Kim. Chain-based fixed-priority scheduling of loosely-dependent tasks. In *ICCD*, pages 631–639. IEEE, 2020.

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

[28] J.-L. Colaço, B. Pagano, and M. Pouzet. Scade 6: A formal language for embedded critical software development. In *Proc. 11th Int. Symp. Theoretical Aspects of Software Engineering (TASE 2017)*, pages 4–15, Nice, France, Sept. 2017. IEEE Computer Society.

[29] D. Dasari, M. Becker, D. Casini, and T. Blaß. End-to-end analysis of event chains under the QNX adaptive partitioning scheduler. In *28th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 214–227, 2022.

[30] D. Dasari, A. Hamann, H. Broede, M. Pressler, and D. Ziegenbein. Brief industry paper: Dissecting the qnx adaptive partitioning scheduler. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 477–480, 2021.

[31] A. Davare, Q. Zhu, M. D. Natale, C. Pinello, S. Kanajan, and A. L. Sangiovanni-Vincentelli. Period optimization for hard real-time distributed automotive systems. In *Design Automation Conference, DAC*, pages 278–283, 2007.

[32] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *Comput. Commun. Rev.*, 36(1):59–62, 2006.

[33] 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 Trans. Embedded Comput. Syst. (Special Issue for CASES)*, 18(5s):58:1–58:24, 2019.

[34] Elektrobit Automotive GmbH. *EB corbos Adaptive AUTOSAR Platform*, 2023.

[35] Elektrobit GmbH. How to execute distributed system-wide cause-effect chains in next-gen cars. Webinar (Tech Corner), June 2020. Hosted on Elektrobit website; accessed 2025-08-06.

[36] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2009.

[37] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti. Scheduling dependent periodic tasks without synchronization mechanisms. In M. Caccamo, editor, *16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2010)*, pages 301–310, Stockholm, Sweden, Apr. 2010. IEEE.

[38] J. Forget, F. Boniol, D. Lesens, and C. Pagetti. A multi-periodic synchronous data-flow language. In *Proc. 11th IEEE High Assurance Systems Engineering Symposium (HASE 2008)*, pages 251–260, Nanjing, China, Dec. 2008. IEEE.

[39] J. Forget, F. Boniol, and C. Pagetti. Verifying end-to-end real-time constraints on multi-periodic models. In *ETFA*, pages 1–8, 2017.

[40] A. Girault, C. Prevot, S. Quinton, R. Henia, and N. Sordon. Improving and estimating the precision of bounds on the worst-case latency of task chains. *IEEE Trans. on CAD of Integrated Circuits and Systems, (Special Issue for EMSOFT)*, 37(11):2578–2589, 2018.

[41] G. Goessler and L. Astefanoaei. Blaming in component-based real-time systems. In *EMSOFT*, pages 7:1–7:10. ACM, 2014.

[42] P. Gohari, M. Nasri, and J. Voeten. Data-age analysis for multi-rate task chains under timing uncertainty. In *RTNS*, pages 24–35. ACM, 2022.

[43] M. Günzel and M. Becker. Optimal task phasing for end-to-end latency in harmonic and semi-harmonic automotive systems. In *RTAS*, pages 164–176. IEEE, 2025.

[44] M. Günzel, K. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J. Chen. Timing analysis of asynchronized distributed cause-effect chains. In *RTAS*, pages 40–52. IEEE, 2021.

[45] M. Günzel, K. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J. Chen. Compositional timing analysis of asynchronized distributed cause-effect chains. *ACM Trans. Embed. Comput. Syst.*, 22(4):63:1–63:34, 2023.

[46] M. Günzel, H. Teper, K. Chen, G. von der Brüggen, and J. Chen. On the equivalence of maximum reaction time and maximum data age for cause-effect chains. In *ECRTS*, volume 262 of *LIPIcs*, pages 10:1–10:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[47] M. Günzel, H. Teper, G. von der Brüggen, and J. Chen. End-to-end latency of cause-effect chains: A tutorial. *ACM Trans. Embed. Comput. Syst.*, 24(1):22:1–22:18, 2025.

[48] M. Günzel, N. Ueter, K. Chen, and J. Chen. Timing analysis of cause-effect chains with heterogeneous communication mechanisms. In *RTNS*, pages 224–234. ACM, 2023.

[49] M. Günzel, N. Ueter, K. Chen, G. von der Brüggen, and J. Chen. Probabilistic reaction time analysis. *ACM Trans. Embed. Comput. Syst.*, 22(5s):143:1–143:22, 2023.

[50] M. Günzel. *Property-based timing analysis of distributed real-time systems*. PhD thesis, TU Dortmund, Germany, 2024. Available at http://dx.doi.org/10.17877/DE290R-24840.

[51] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proc. IEEE*, 79(9):1305–1320, Sept. 1991.

[52] 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, ECRTS*, pages 10:1–10:20, 2017.

[53] C. Hong, M. Caesar, and B. Godfrey. Finishing flows quickly with preemptive scheduling. In *SIGCOMM*, pages 127–138. ACM, 2012.

[54] G. Iooss, A. Cohen, D. Potop-Butucaru, M. Pouzet, V. Bregeon, J. Souyris, and P. Baufreton. Polyhedral scheduling and relaxation of synchronous reactive systems. In *12th Int. Workshop on Polyhedral Compilation Techniques (IMPACT 2022)*, Budapest, Hungary, June 2022. HiPEAC.

[55] X. Jiang, D. Ji, N. Guan, R. Li, Y. Tang, and W. Yi. Real-time scheduling and analysis of processing chains on multi-threaded executor in ROS 2. In *RTSS*, pages 27–39. IEEE, 2022.

[56] 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. In *DATE*, pages 1–6. IEEE, 2023.

[57] S. Johannessen. Time synchronization in a local area network. *IEEE Control Systems Magazine*, pages 61–69, 2004. About NTP and IEEE 1588.

[58] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Proc. 6th Int. Federation for Information Processing (IFIP) Congress 1974*, pages 471–475, Stockholm, Sweden, Aug. 1974. North-Holland.

[59] S. Kim, J. Song, K. Lee, S. Oh, and H. S. Chwa. Cros-Rt: Cross-Layer Priority Scheduling for Predictable Inter-Process Communication in Ros 2 . In *2025 IEEE 31st Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 202–214, Los Alamitos, CA, USA, May 2025. IEEE Computer Society.

[60] T. Klaus, M. Becker, W. Schröder-Preikschat, and P. Ulbrich. Constrained data-age with job-level dependencies: How to reconcile tight bounds and overheads. In *27th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 66–79, 2021.

[61] T. Klaus, F. Franzmann, M. Becker, and P. Ulbrich. Data propagation delay constraints in multi-rate systems: Deadlines vs. job-level dependencies. In *RTNS*, pages 93–103. ACM, 2018.

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

[63] T. Kloda, J. Chen, A. Bertout, L. Sha, and M. Caccamo. Latency analysis of self-suspending task chains. In *DATE*, pages 1299–1304. IEEE, 2022.

[64] S. Laddad, C. Power, M. Milano, A. Cheung, N. Crooks, and J. M. Hellerstein. Keep CALM and CRDT on. *arXiv*, 2210.12605v1 [cs.DB], 2022.

[65] E. A. Lee. Logical time in actor systems. In J. Meseguer, C. A. Varela, and N. Venkatasubramanian, editors, *Concurrent Programming, Open Systems and Formal Methods: Essays Dedicated to Prof. Gul Agha to Celebrate his Scientific Career*, volume LNCS, to appear. Springer, 2025.

[66] E. A. Lee, R. Akella, S. Bateni, S. Lin, M. Lohstroh, and C. Menard. Consistency vs. availability in distributed cyber-physical systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 22(5s):1–24, 2023. Presented at EMSOFT, September 17-22, 2023, Hamburg, Germany.

[67] E. A. Lee and H. Zheng. Operational semantics of hybrid systems. In M. Morari and L. Thiele, editors, *Proc. 8th Int. Workshop on Hybrid Systems: Computation and Control (HSCC 2005)*, number 3414 in LNCS, pages 25–53, Zurich, Switzerland, Mar. 2005. Springer.

[68] E. A. Lee and H. Zheng. Leveraging synchronous language principles for heterogeneous modeling and design of embedded systems. In C. M. Kirsch and R. Wilhelm, editors, *Proc. 7th ACM Int. Conf. on Embedded Software (EMSOFT 2007)*, pages 114–123, Salzburg, Austria, Sept. 2007. ACM Press.

[69] H. Lee, Y. Choi, T. Han, and K. Kim. Probabilistically guaranteeing end-to-end latencies in autonomous vehicle computing systems. *IEEE Trans. Computers*, 71(12):3361–3374, 2022.

[70] S. Li and E. A. Lee. A preliminary model of coordination-free consistency. *arXiv*, 2504.01141 [cs.DC], 2025.

[71] L. Lo Bello and W. Steiner. A perspective on ieee time-sensitive networking for industrial communication and automation systems. *Proceedings of the IEEE*, 107(6):1094–1120, 2019.

[72] 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.

[73] M. Lohstroh, C. Menard, S. Bateni, and E. A. Lee. Toward a lingua franca for deterministic concurrent systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(4):Article 36, 2021.

[74] L. Maia and G. Fohler. Reducing end-to-end latencies of multi-rate cause-effect chains in safety critical embedded systems. In *12th European Congress on Embedded Real Time Software and Systems (ERTS 2024)*, 2024.

[75] J. Martinez, I. S. Olmedo, and M. Bertogna. Analytical characterization of end-to-end communication delays with logical execution time. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 37(11):2244–2254, 2018.

[76] F. Paladino, A. Biondi, E. Bini, and P. Pazzaglia. Optimizing per-core priorities to minimize end-to-end latencies. In *ECRTS*, volume 298 of *LIPIcs*, pages 6:1–6:25. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

[77] F. Paladino, A. Biondi, E. Bini, and P. Pazzaglia. Optimizing Per-Core Priorities to Minimize End-To-End Latencies. In R. Pellizzoni, editor, *36th Euromicro Conference on Real-Time Systems (ECRTS 2024)*, volume 298 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:25, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[78] P. Pazzaglia and M. Maggio. Characterizing the effect of deadline misses on time-triggered task chains. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 41(11):3957–3968, 2022.

[79] PerceptIn. RTSS industry challenge 2021. In *Real-Time Systems Symposium (RTSS)*, 2021.

[80] A. C. Rajeev, S. Mohalik, M. G. Dixit, D. B. Chokshi, and S. Ramesh. Schedulability and end-to-end latency in distributed ECU networks: formal modeling and precise estimation. In *EMSOFT*, pages 129–138. ACM, 2010.

[81] A. Ramesh, T. Huang, E. Ruppel, D. Dasari, B. Pourmohseni, F. Smirnov, M. Giani, P. Pazzaglia, C. Shelton, N. Pereira, A. Hamann, D. Ziegenbein, and A. Rowe. Silverline: Lightweight virtualization and orchestration of distributed systems. In *31st IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 375–388, 2025.

[82] J. Schlatow and R. Ernst. Response-time analysis for task chains in communicating threads. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 245–254, 2016.

[83] J. Schlatow, M. Möstl, S. Tobuschat, T. Ishigooka, and R. Ernst. Data-age analysis and optimisation for cause-effect chains in automotive control systems. In *IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–9, 2018.

[84] G. Sciangula, D. Casini, A. Biondi, C. Scordino, and M. Di Natale. Bounding the Data-Delivery Latency of DDS Messages in Real-Time Applications. In A. V. Papadopoulos, editor, *35th Euromicro Conference on Real-Time Systems (ECRTS 2023)*, volume 262 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:26, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[85] X. Shi, Y. Zhu, and L. Li. End-to-end probability analysis method for multi-core distributed systems. *J. Supercomput.*, 80(19):26751–26775, 2024.

[86] H. Sobhani, H. Choi, and H. Kim. Timing analysis and priority-driven enhancements of ROS 2 multi-threaded executors. In *RTAS*, pages 106–118. IEEE, 2023.

[87] C. Sofronis, S. Tripakis, and P. Caspi. A memory-optimal buffering protocol for preservation of synchronous semantics under preemptive scheduling. In S. L. Min and Y. Wang, editors, *Proc. 6th ACM Int. Conf. on Embedded Software (EMSOFT 2006)*, pages 21–33, Seoul, South Korea, Oct. 2006. ACM Press.

[88] Y. Tang, Z. Feng, N. Guan, X. Jiang, M. Lv, Q. Deng, and W. Yi. Response time analysis and priority assignment of processing chains on ROS2 executors. In *RTSS*, pages 231–243. IEEE, 2020.

[89] H. Teper, T. Betz, M. Günzel, D. Ebner, G. von der Brüggen, J. Betz, and J. Chen. End-to-end timing analysis and optimization of multi-executor ROS 2 systems. In *RTAS*, pages 212–224. IEEE, 2024.

[90] H. Teper, M. Günzel, N. Ueter, G. von der Brüggen, and J. Chen. End-to-end timing analysis in ROS2. In *IEEE Real-Time Systems Symposium, RTSS*, pages 53–65, 2022.

[91] The MathWorks Inc. *Simulink version: 24.2 (R2024b)*. Natick, MA, USA, Sept. 2024.

[92] TTTech Auto. *MotionWise Safety Middleware for SDVs*, 2023.

[93] S. Wang, D. Li, S. Huang, X. Deng, A. H. Sifat, J. Huang, C. Jung, R. K. Williams, and H. Zeng. Time-triggered scheduling for nonpreemptive real-time DAG tasks using 1-opt local search. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 43(11):3650–3661, 2024.

[94] S. Wang, D. Li, A. H. Sifat, S.-Y. Huang, X. Deng, C. Jung, R. Williams, and H. Zeng. Optimizing Logical Execution Time Model for Both Determinism and Low Latency. In *30th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 135–148, 2024.

[95] R. Wyss, F. Boniol, J. Forget, and C. Pagetti. A synchronous language with partial delay specification for real-time systems programming. In R. Jhala and A. Igarashi, editors, *Proc. 10th Asian Symp. Programming Languages and Systems (APLAS 2012)*, volume 7705 of *LNCS*, pages 223–238, Kyoto, Japan, Dec. 2012. Springer.

[96] R. Wyss, F. Boniol, C. Pagetti, and J. Forget. End-to-end latency computation in a multi-periodic design. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC*, pages 1682–1687, 2013.

[97] Z. Zhang, Z. Li, H. Kim, and C. Liu. BOXR: body and head motion optimization framework for extended reality. In *IEEE Real-Time Systems Symposium, RTSS*, pages 70–82, 2024.