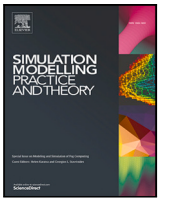
Simulation Modelling Practice and Theory 152 (2026) 103326

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat



Step-consistent high-throughput shared-memory communication for FMI-based co-simulation

Woo-Sung Jung *, Jeongsik Kim, Ho-Min Park, Saehoon Kang, Dae Seung Yoo

Electronics and Telecommunications Research Institute, Republic of Korea

ARTICLE INFO

Keywords:
FMI 2.0
Co-simulation
Shared memory
Simulation interoperability
Deterministic execution
Real-time simulation
Digital twin

ABSTRACT

High-throughput data exchange is a critical requirement for FMI-based co-simulation, particularly in digital twin and cyber–physical system validation scenarios involving large shared state variables. Although shared-memory communication can significantly reduce latency by eliminating redundant memory copies, naïve implementations often violate FMI step semantics when multiple consumers concurrently access shared buffers. Such violations can lead to step mismatches, torn reads, and non-deterministic simulation behavior even when wall-clock timing constraints are satisfied.

This paper proposes a simulation-time-aware shared-memory communication architecture for FMI 2.0 co-simulation. The proposed design separates bulk data transfer from simulation-step coordination by introducing a dual-path mechanism consisting of high-bandwidth shared memory for payload exchange and a lightweight step-indexed synchronization channel coordinated by the FMI master. By preserving previous-step data until all consumers acknowledge completion, the architecture guarantees deterministic step-consistent visibility across multiple consumers.

Extensive experiments under both clean and stress execution conditions demonstrate that relaxed shared-memory schemes achieve lower raw latency and higher throughput but suffer from step mismatches and data integrity violations. In contrast, the proposed architecture achieves zero step mismatches and zero CRC failures across all evaluated payload sizes while maintaining competitive latency scaling and stable throughput in broadcast configurations. These results highlight the importance of integrating simulation-time semantics into communication-layer design and show that the proposed approach provides a practical and FMI-compliant solution for deterministic high-throughput co-simulation.

1. Introduction

Digital transformation has become a central paradigm across a wide range of engineering domains, including autonomous vehicles, smart manufacturing, maritime systems, and large-scale cyber–physical infrastructures. As these systems grow in complexity and connectivity, simulation has evolved from an offline design tool into a core capability for system validation, verification, and operational decision support. In particular, co-simulation enables heterogeneous models — such as physical dynamics, control logic, communication networks, and artificial intelligence components — to interact within a unified execution framework.

* Corresponding author.
E-mail address: woosung@etri.re.kr (W.-S. Jung).

https://doi.org/10.1016/j.simpat.2026.103326
Received 29 April 2026; Received in revised form 4 July 2026; Accepted 17 July 2026
Available online 23 July 2026

Table 1
Comparison of related work with respect to FMI compatibility, large-data support, and communication-layer properties.

Category reference	FMI 2.0	FMI 3.0	Large data	FMU buffering	Data integrity	Step latency	Throughput	Deterministic execution
Blochwitz et al. [3,4]	✓	–	–	–	–	–	–	–
FMI 2.0 Spec. [5]	✓	–	–	–	–	–	–	–
FMI 3.0 Spec. [6–8]	–	✓	✓	Partial	Partial	Partial	✓	Partial
Gomes et al. [9]	✓	✓	Partial	–	Partial	Partial	Partial	Partial
Schweiger et al. [10]	✓	Partial	Partial	–	–	Partial	–	Partial
Palensky et al. [11]	✓	–	Partial	–	–	Partial	–	✓
INTO-CPS/Maestro [12,13]	✓	–	Partial	–	–	–	Partial	–
OMSimulator [14]	✓	–	Partial	–	–	–	Partial	–
NeuralFMU [15,16]	✓	Partial	✓	–	–	–	✓	–
IPC/Double Buffering [17]	–	–	✓	✓	–	✓	✓	–
This work	✓	–	✓	✓	✓	✓	✓	✓

The Functional Mock-up Interface (FMI) has emerged as a widely adopted standard for enabling such heterogeneous co-simulation. By encapsulating simulation models as Functional Mock-up Units (FMUs) and coordinating them through a master algorithm, FMI enables tool-independent integration of models developed across different engineering domains. Consequently, FMI has been adopted in a broad range of industrial applications, including energy systems, automotive engineering, maritime systems, and logistics simulations.

Recent trends in digital twins, autonomous operation, and data-driven modeling place unprecedented demands on FMI-based co-simulation environments. Modern simulation scenarios increasingly involve large and high-frequency data streams, such as camera images, radar point clouds, sensor arrays, and machine-learning inference outputs. Under such conditions, the traditional FMI data exchange mechanism based on variable-centric API calls (e.g., `fmiGetXXX` and `fmiSetXXX`) becomes a performance bottleneck due to repeated memory copies and centralized coordination through the master algorithm.

To address these limitations, FMI 3.0 introduces several extensions, including binary variables, clock-based execution, and scheduled synchronization. These features provide more expressive interfaces for data exchange and timing control and potentially improve performance in data-intensive scenarios. However, the practical adoption of FMI 3.0 remains limited in many industrial environments. A large body of existing simulation assets, toolchains, and certification workflows remains tightly coupled to FMI 2.0, making migration costly due to redevelopment, revalidation, and tool compatibility requirements.

This statement does not imply that FMI 3.0 is unsuitable for data-intensive co-simulation. Rather, the present work targets environments where FMI 2.0 assets and toolchains must be preserved. An empirical comparison with FMI 3.0 binary variables and clock-based execution is outside the scope of this paper.

Moreover, even when binary variables are available, performance limitations often originate from the underlying data transfer path rather than from the FMI interface itself. Repeated data copies mediated by the master algorithm can introduce memory bandwidth overhead, synchronization delays, and contention when multiple FMUs exchange large payloads at high update rates. This observation motivates the proposed FMI 2.0-compatible shared-memory data-path design. Shared-memory and IPC mechanisms have been widely studied as efficient communication techniques for reducing copy overhead in tightly coupled computing environments [1,2]. However, such mechanisms do not inherently provide simulation-step semantics or FMI-compliant visibility guarantees. Therefore, when shared memory is used outside the standard FMI API, additional coordination is required to ensure that data produced at a given logical simulation time are consumed consistently by all participating FMUs.

This paper addresses the communication-layer design problem of FMI 2.0-based co-simulation under data-intensive workloads. We propose an FMI-compatible dual-path communication architecture that separates simulation control semantics from bulk data transfer. The proposed approach introduces a high-throughput shared-memory data plane for payload exchange while maintaining step-consistent synchronization through the FMI master.

By decoupling large data transfers from the conventional FMI API while preserving simulation-step semantics, the architecture enables low-latency, high-throughput, and deterministic co-simulation. Compared with controlled relaxed shared-memory baselines that prioritize raw transfer performance without enforcing FMI step semantics, the proposed design guarantees step-consistent visibility across multiple consumers while maintaining competitive latency scaling.

The main contributions of this paper are summarized as follows:

- We identify the communication-layer bottleneck of FMI 2.0-based co-simulation in data-intensive scenarios and analyze the trade-off between raw data transfer performance and step-consistent execution semantics.
- We propose an FMI-compatible dual-path communication architecture that combines high-throughput shared-memory data transfer with simulation-time-aware synchronization and integrity protection.
- We experimentally demonstrate that the proposed architecture expands the feasible operating region for deadline-constrained co-simulation while guaranteeing zero step mismatches and zero CRC failures under stress conditions.

These contributions provide a practical approach for improving single-host, shared-memory-based FMI 2.0 co-simulation under data-intensive workloads without requiring migration to FMI 3.0 or modifications to existing master algorithms. Distributed transport, clock synchronization drift, and partial FMU failures are outside the scope of the present evaluation and are discussed as future work.

2. Related work

Table 1 summarizes representative studies and compares their coverage with respect to key communication-layer requirements relevant to FMI-based co-simulation.

2.1. FMI standards and co-simulation semantics

The Functional Mock-up Interface (FMI) has become the de facto standard for tool-independent model exchange and co-simulation, enabling heterogeneous simulation components to interoperate through a common interface [3,4]. In FMI-based co-simulation, simulation models are encapsulated as Functional Mock-up Units (FMUs), while a master algorithm coordinates time advancement and data exchange among participating components [5].

In FMI 2.0, data exchange is realized primarily through variable-oriented API calls such as `fmiGetXXX` and `fmiSetXXX`. This mechanism provides clear synchronization semantics and broad interoperability, but it can become a bottleneck in data-intensive scenarios involving large payloads or high update frequencies. To address some of these limitations, FMI 3.0 extends the standard with binary variables, clocks, and scheduled execution [6–8]. These features improve interface expressiveness and timing control, but migration to FMI 3.0 still requires substantial redevelopment and revalidation in many industrial environments.

For this reason, FMI 2.0 remains dominant in practical co-simulation workflows, especially where legacy FMUs, mature toolchains, and certification-oriented processes must be preserved [9,10]. Accordingly, improving communication efficiency while retaining FMI 2.0 semantics remains a practically relevant research problem.

2.2. Large-scale and data-intensive co-simulation

Several frameworks have been developed to orchestrate large-scale FMI-based co-simulation scenarios. INTO-CPS and Maestro provide model orchestration, scenario management, and tool integration for complex cyber–physical systems [12,13]. Similarly, OMSimulator supports composite modeling, SSP-based workflows, and co-simulation integration across FMI-based components [14].

In smart-grid and energy domains, FMI is often used together with orchestration environments such as mosaik to coordinate distributed simulators and heterogeneous models [18–20]. These studies demonstrate the scalability and flexibility of FMI-based ecosystems, but also report increasing overhead when models exchange data at high rates or in large volumes.

Recent efforts further extend FMI-based co-simulation toward data-intensive digital twin and learning-enabled applications. NeuralFMU integrates FMUs into machine-learning workflows [15,16], while CoFMPy targets rapid prototyping of FMI-based digital twins [21]. Other studies explore FMI-based coupling of simulation tools and PDE solvers for multi-physics systems [22,23]. Taken together, these works highlight the growing importance of efficient communication mechanisms for FMI-based environments, particularly when large data objects must be exchanged repeatedly during simulation.

2.3. Buffering and shared-memory communication

Efficient interprocess communication mechanisms such as shared memory, user-level IPC, and double buffering have long been studied in operating systems and concurrent systems research [1,17]. Lamport introduced fundamental principles for interprocess communication in concurrent systems, while Bershad et al. demonstrated that user-level IPC can substantially reduce communication overhead on shared-memory multiprocessors. These studies established the basis for reducing kernel-mediated communication costs and avoiding unnecessary memory copies in tightly coupled execution environments.

Within FMI ecosystems, however, buffering-related studies have mostly focused on FMU handling, tool integration, or internal optimization rather than explicit FMU-to-FMU data-plane design. Libraries such as FMI++ simplify FMU wrapping and orchestration but do not define shared-memory communication structures across FMUs [24]. Similarly, universal FMU approaches and FMI-based coupling runners emphasize interoperability and workflow integration rather than deterministic high-throughput transport between concurrently executing FMUs [22,25].

Shared-memory IPC has also been explored in co-simulation contexts outside general-purpose FMI data exchange. For example, Scheibe et al. proposed a shared-memory IPC-based co-simulation concept for power system analysis, demonstrating the potential of shared memory to improve communication efficiency between coupled simulation components [2]. In parallel, Bastian et al. discussed master algorithms for FMI-based co-simulation, providing an important foundation for understanding how time advancement and data exchange are coordinated in FMI environments [26].

As a result, generic IPC mechanisms cannot be directly assumed to preserve simulation-step semantics in FMI-based co-simulation. Generic IPC designs primarily optimize data transfer latency or throughput, whereas FMI-based co-simulation additionally requires simulation-step-consistent visibility, deterministic data consumption, and compatibility with master-driven time advancement. When applied naively, shared-memory communication can lead to overwrite hazards, inconsistent visibility, and temporal mismatches between producers and consumers, especially under asynchronous execution and broadcast-oriented workloads. This distinction motivates the step-indexed synchronization mechanism proposed in this work.

2.4. Synchronization and data integrity

Synchronization has long been recognized as a core challenge in co-simulation, particularly in cyber–physical and energy systems where heterogeneous time scales and solvers interact [9,11]. Most prior work addresses this issue at the level of master algorithms, scheduling strategies, or numerical coupling policies. By contrast, fewer studies consider synchronization directly at the communication layer where large data objects are exchanged.

Related concerns arise in memory protection and integrity verification. Memory protection mechanisms such as protection keys provide lightweight control over access permissions in concurrent execution contexts [27]. CRC and checksum mechanisms are similarly well established for detecting corruption in communication and embedded systems [28,29]. However, these mechanisms are usually treated independently from co-simulation timing semantics. In this paper, integrity refers to error detection for torn reads, partially written payloads, or inconsistent shared-memory visibility. It does not imply cryptographic protection or security against malicious or compromised FMUs.

In FMI-based co-simulation, step consistency, concurrent visibility, and integrity verification are tightly coupled when large payloads are exchanged outside the standard FMI API. This makes it necessary to consider synchronization, integrity, and access control together rather than as separate design concerns.

2.5. Research gap and baseline design rationale

The literature reviewed above reveals two largely independent lines of work. FMI-based studies focus primarily on interoperability, orchestration, and heterogeneous system integration [9,12,14]. In contrast, IPC and shared-memory studies focus on efficient data transfer and buffering strategies [1,2,17], often without considering simulation-time semantics. The gap addressed in this paper is therefore not the use of shared memory itself, but the integration of shared-memory communication with FMI 2.0 step semantics. Existing IPC mechanisms provide efficient data movement, but they do not define how data validity should be bound to FMI simulation steps or how multiple FMUs should observe the same logical-time data consistently.

However, few existing studies address how high-throughput communication can be integrated into FMI 2.0-based co-simulation while simultaneously satisfying four practical requirements: (i) compatibility with FMI 2.0 execution semantics, (ii) efficient transfer of large payloads, (iii) simulation-step-consistent delivery, and (iv) integrity-preserving concurrent access.

This gap directly motivates the controlled baseline configurations used in this paper. The evaluated schemes are not intended to represent complete existing software frameworks; rather, they are author-implemented communication counterparts designed to isolate the effects of data-path choice and step-semantics enforcement under the same FMI 2.0 master, workload, and measurement infrastructure. Baseline 1 (B1) follows the conventional FMI 2.0 API-mediated communication model, in which data exchange is performed through the FMI interface and coordinated by the master algorithm. Baseline 2-r (B2-r) is an author-implemented single-buffer shared-memory counterpart inspired by generic producer–consumer IPC designs, where a producer may overwrite the latest value without waiting for all consumers. Baseline 2-g (B2-g) is an author-implemented guarded variant of B2-r that adds a minimal step-stamp check at the consumer side to evaluate how far simple guard-based protection can restore simulation-step consistency.

The proposed method (P) fills this gap by combining shared-memory bulk transfer with ping–pong buffering, step-indexed acknowledgments, and simulation-time-aware integrity checks while preserving FMI 2.0 execution semantics.

3. Proposed architecture

This section presents a dual-path communication architecture that enhances data exchange performance in FMI-based co-simulation while preserving FMI 2.0 execution semantics.

Fig. 1 illustrates the overall structure of the proposed dual-path communication architecture. The design separates simulation control from bulk data transfer by introducing two independent communication paths. The control path follows the standard FMI execution model, where the FMI master coordinates simulation steps and lifecycle management through FMI API calls such as `fmiDoStep`. In contrast, large payload data are exchanged directly between FMUs through a shared-memory data path.

Each FMU exposes two logical interfaces: a control port and a data port. The control port interacts with the FMI master using the conventional FMI API, while the data port accesses a dedicated shared-memory region assigned to the FMU. Within the shared-memory pool, each FMU is allocated a statically defined buffer region with ping–pong buffering to support safe concurrent access.

To preserve deterministic execution, each buffer maintains lightweight metadata including a timestamp, status flags, and an optional CRC checksum. These metadata fields enable step-level visibility control and integrity verification while avoiding repeated memory copies through the FMI API.

At each `fmiDoStep` invocation, the FMI master first determines the logical simulation step to be executed and then coordinates the visibility of shared-memory buffers associated with that step. The producer FMU writes the next-step payload to the inactive buffer, while consumer FMUs continue to read the previously committed buffer until the master exposes the new step-indexed buffer. This sequence ensures that high-throughput data transfer occurs outside the FMI API, while logical-time visibility remains synchronized with the FMI master lifecycle.

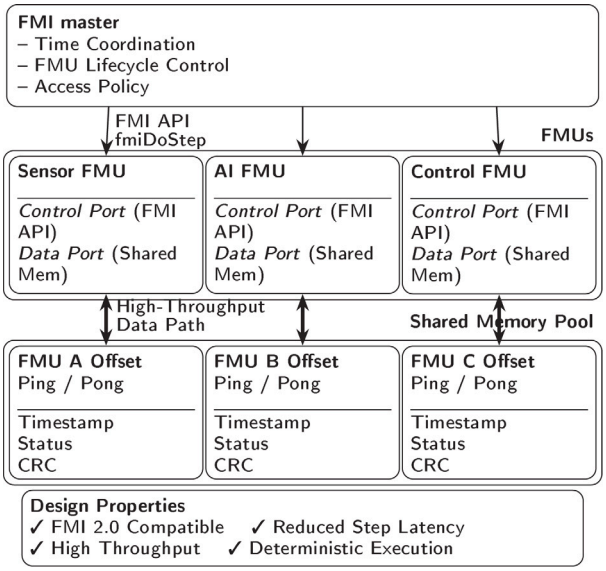


Fig. 1. Dual-path shared-memory communication architecture for FMI 2.0-based co-simulation.

3.1. Architecture overview and design rationale

The proposed architecture is motivated by the observation that large-payload communication and simulation-step coordination impose fundamentally different requirements in FMI-based co-simulation. Conventional FMI interactions are well suited for lifecycle management, time stepping, and scalar control exchange, but repeated `fmiGet/fmiSet` calls become inefficient when large data objects must be transferred at high frequency.

To address this mismatch, the proposed design preserves the FMI API for simulation control while introducing a separate high-throughput shared-memory path for bulk data exchange. This separation allows large payloads to bypass the master-mediated copy path without changing the FMI execution model. As a result, the architecture improves communication efficiency while remaining compatible with existing FMI 2.0 master algorithms and FMUs.

3.2. FMU-level shared memory with double buffering

The proposed data path is realized through a shared memory pool that is partitioned at the FMU level. Each FMU is assigned a statically defined memory region with a fixed offset that remains unchanged throughout the simulation. This FMU-level partitioning establishes clear ownership boundaries and prevents unintended memory access among concurrent FMUs.

Within each FMU-assigned region, a ping–pong (double-buffered) structure is employed. One buffer is used exclusively for data production, while the other buffer is available for data consumption. Buffer roles are switched only after a complete data update is committed, eliminating read–write conflicts without introducing blocking synchronization primitives. The proposed ping–pong structure trades additional memory usage for deterministic visibility. For a payload of size S, a single-buffer shared-memory design requires approximately S bytes per shared data object, whereas the proposed double-buffer design requires approximately $2S + M$ bytes, where M denotes lightweight metadata such as the step index, status flags, CRC value, and acknowledgment state. For the evaluated payload sizes of 64 KB, 256 KB, 1 MB, and 4 MB, the corresponding double-buffer payload memory becomes approximately 128 KB, 512 KB, 2 MB, and 8 MB, respectively, excluding the small metadata overhead. Thus, the dominant memory overhead comes from the second payload buffer, while the metadata overhead remains comparatively small.

This structure enables concurrent execution of producer and consumer FMUs and supports atomic transfer of large payloads, such as sensor arrays, perception outputs, or machine-learning inference results. Unlike generic IPC mechanisms, the proposed buffering scheme is explicitly aligned with FMU execution boundaries and simulation step progression.

3.3. Step-indexed synchronization and integrity control

To ensure deterministic behavior, each buffer pair is accompanied by lightweight metadata that include a simulation-time timestamp, buffer state flags, and an optional integrity check value. The timestamp associates each data update with a specific simulation step, allowing receiving FMUs to verify temporal consistency independently of wall-clock execution order. In the

Algorithm 1 Step-indexed double-buffer synchronization

Require: Simulation step k, producer FMU F_p, consumer set C, buffers B^{ping} and B^{pong}
$B_w \leftarrow$ inactive buffer not currently visible to consumers
$B_r \leftarrow$ buffer committed for the previous visible step

Producer-side update
F_p writes payload for step k into B_w
Compute metadata $m_k \leftarrow \{step = k, status = \textsc{Writing}, crc = \mathrm{CRC}(B_w)\}$
Atomically set $m_k.status \leftarrow \textsc{Committed}$
Submit commit notification (k, B_w, m_k) to the FMI master

Master-side visibility control
if all consumers have acknowledged completion of step $k-1$ **then**
    Expose B_w as the visible buffer for step k
    Reset acknowledgments for all $c \in C$
else
    Preserve B_r as the visible buffer until pending acks are received
end if

Consumer-side read and acknowledgment
for all $c \in C$ **do**
    c reads metadata of the currently visible buffer
    if $m.step = k$ and $m.status = \textsc{Committed}$ **then**
        c reads the payload from the visible buffer
        Validate $\mathrm{CRC}(payload) = m.crc$
        Send $\mathrm{ack}(c, k)$ to the FMI master
    else
        Defer consumption until the expected step becomes visible
    end if
end for

implementation, the timestamp is represented as a monotonically increasing simulation-step index assigned by the FMI master. Each shared-memory entry maintains two payload buffers, denoted as `ping` and `pong`, together with metadata fields for the committed step index, write status, CRC value, and per-consumer acknowledgment state. Only the committed buffer is visible to consumers, while the inactive buffer is used for the next producer update.

Algorithm 1 summarizes the proposed step-indexed double-buffer synchronization procedure. The producer never overwrites the buffer currently visible to consumers; instead, it writes the next-step payload into the inactive buffer and commits the buffer only after the payload and metadata have been completely updated. The FMI master then controls when the newly committed buffer becomes visible. If any consumer has not acknowledged the previous step, the previous visible buffer is preserved, preventing consumers from observing partially updated or future-step data. Each consumer validates both the simulation-step index and the CRC before acknowledging completion.

This procedure differs from the baseline schemes in the handling of visibility and overwrite protection. In B1, step consistency is preserved by the standard FMI API and master-mediated data exchange, but large payloads are repeatedly copied through the conventional communication path. In B2-r, the producer writes to a single shared buffer without consumer acknowledgments or step-indexed visibility control; therefore, a new payload may overwrite data that some consumers have not yet read. In B2-g, consumers check whether the buffer step stamp matches the expected simulation step, but the underlying single-buffer overwrite policy remains unchanged, which can introduce waiting and straggler effects under broadcast workloads. In contrast, the proposed method combines ping–pong buffering, step-indexed metadata, CRC validation, and per-consumer acknowledgments to ensure that all consumers observe a consistent logical-time view of the shared payload.

The FMI master retains global authority over synchronization and access control. During initialization, it assigns FMU identifiers, configures memory offsets, and establishes access permissions. During execution, it ensures that shared-memory visibility remains consistent with simulation-time advancement.

By binding data validity to simulation time rather than execution order, the proposed architecture maintains deterministic behavior even under asynchronous FMU execution. The CRC-based integrity check is used to detect torn reads, partially written payloads, or corrupted shared-memory data during FMU-to-FMU data exchange. It is not intended to provide cryptographic authentication or protection against malicious or compromised FMUs. Thus, the integrity mechanism considered in this paper should be interpreted as communication-level error detection rather than a security mechanism.

Table 2
Experimental platform.

Item	Specification
Host CPU	Intel i9-12900KF
Execution environment	Virtualized environment
Allocated CPU cores	8 vCPUs
Memory	32 GB
Operating system	Ubuntu 24.04 LTS
Kernel version	Linux 6.17.0
Compiler	gcc 13.3.0
Shared-memory backend	mmap
Timer source	C++ `std::chrono::steady_clock`

4. Experimental setup

This section describes the experimental environment, evaluation criteria, and baseline configurations used to assess the proposed dual-path FMU communication architecture. The objective is to provide a fair and reproducible comparison under data-intensive co-simulation workloads, focusing on latency, throughput, and execution determinism.

4.1. Evaluation metrics

To assess the effectiveness of the proposed architecture in data-intensive co-simulation scenarios, we consider the following three key performance indicators (KPIs), which jointly capture performance, scalability, and correctness aspects that are not simultaneously addressed in existing FMI-based co-simulation studies.

Step Latency: Step latency measures the elapsed wall-clock time required to complete a single simulation step, including data exchange and synchronization among FMUs. This metric captures the impact of communication overhead on simulation progress and is particularly relevant for real-time or near-real-time co-simulation scenarios.

Throughput: Throughput represents the sustained data transfer rate between FMUs during simulation execution. It reflects the ability of the communication mechanism to handle large payloads and high update frequencies, such as sensor streams or machine-learning inference outputs.

Determinism: Determinism evaluates the consistency of simulation outcomes across repeated executions under identical initial conditions. Specifically, it measures whether data are consumed at the correct simulation steps and whether execution order variations lead to divergent results. Deterministic behavior is essential for reproducibility, debugging, and certification-oriented simulation workflows.

4.2. Experimental configuration

The experimental setup consists of a co-simulation scenario involving multiple FMUs representing heterogeneous functional roles. In the evaluated configuration, a producer FMU generates large data payloads at each simulation step, while one or more consumer FMUs process the data and optionally feed results back to a control FMU.

All FMUs are executed under an FMI 2.0-compatible master algorithm. The proposed shared-memory communication mechanism is enabled alongside the standard FMI control path, while baseline configurations rely on conventional `fmiGet/fmiSet` calls for data exchange. The simulation step size is fixed across all experiments to ensure a fair comparison.

All experiments were conducted on a single workstation to isolate communication-layer behavior without introducing variability from distributed network transport. The experiments were executed in a virtualized environment configured with an 8-vCPU allocation on the Intel i9-12900KF host CPU. Accordingly, the core count reported in Table 2 denotes the CPU resources allocated to the experimental environment rather than the total physical core count of the host processor. Table 2 summarizes the hardware and software environment used for the evaluation. The platform information is reported because step latency and tail behavior can be affected by CPU architecture, memory hierarchy, operating-system scheduling, and timer resolution.

To evaluate scalability, experiments are conducted with varying payload sizes and data update frequencies. Each experiment is repeated multiple times to assess variability and determinism. Wall-clock execution time, data transfer statistics, and simulation-step alignment are recorded for analysis.

Table 3 summarizes the key experimental workload parameters used throughout the evaluation. The platform configuration is summarized separately in Table 2, while Table 3 reports the simulation parameters varied across experiments. The selected ranges are intended to reflect realistic data-intensive co-simulation workloads, where a single producer FMU broadcasts data to multiple consumer FMUs at varying update rates.

The number of consumers (N) is varied to evaluate scalability under broadcast communication. Payload sizes range from tens of kilobytes to several megabytes to cover both lightweight state updates and large sensor-like data streams. Update frequencies between 20 Hz and 200 Hz are considered to assess both near-real-time and high-frequency execution regimes. The stress condition was implemented as a single-host process-level experiment comprising one producer process and 16 independently launched consumer processes for each communication scheme. After completing its consumer-side processing for an observed step, each

Table 3
Experimental parameters.

Parameter	Values
Topology	1:N broadcast (N = 1, 2, 4, 8, 16)
Payload size	64 KB, 256 KB, 1 MB, 4 MB
Update frequency	20 Hz, 50 Hz, 100 Hz, 200 Hz
Execution mode	Clean/Stress
FMU placement	Separate OS processes
Repetitions	50 independent runs

consumer invoked `std::this_thread::sleep_for` for $D+U$ microseconds, where the archived stress runs used $D = 1000$ μs and $U \sim \mathcal{U}_{\text{int}}\{0, \ldots, 300\}$ μs, yielding a per-observation delay of 1000–1300 μs. The same delay configuration was applied to B1, B2-r, B2-g, and P, and no external CPU-load generator, OS-level scheduling-priority perturbation, distributed network delay, clock drift, or remote-FMU failure was injected. All FMUs are executed as separate operating-system processes to capture scheduling effects and synchronization overheads encountered in practical deployments. Each configuration is repeated 50 times with 1000 simulation steps per run to evaluate variability and execution determinism.

4.3. Baseline configurations

To evaluate the proposed architecture, we compare it against baseline configurations representing the major communication strategies discussed in the literature. Specifically, we contrast the conventional FMI 2.0 API-based communication model with shared-memory communication approaches inspired by classical IPC designs. All baseline schemes were implemented by the authors within the same experimental framework to ensure a controlled and reproducible comparison. B1 implements the standard FMI 2.0 API-based data exchange pattern. B2-r and B2-g are author-defined shared-memory counterparts inspired by generic IPC producer–consumer communication patterns rather than direct reproductions of a particular published software implementation. Their purpose is to expose the semantic consequences of removing or minimally restoring step-level protection under identical workload and execution conditions.

The FMI API baseline reflects the standard-compliant communication model in which all data exchanges between FMUs are performed through FMI 2.0 function calls. This configuration represents the widely adopted workflow in FMI-based co-simulation and therefore serves as the primary reference for correctness-preserving communication.

In contrast, shared-memory communication can significantly improve raw data-transfer performance by avoiding repeated data copies. However, such approaches typically lack explicit awareness of simulation-step semantics, which may lead to inconsistent data visibility or overwrite hazards under concurrent execution.

To examine this trade-off, we evaluate two variants of a single-buffer shared-memory baseline.

B2-r (Relaxed Single-Buffer SHM) employs a single shared-memory buffer without enforcing step-level synchronization. Producers overwrite the buffer whenever new data become available, following a latest-wins policy. This design minimizes synchronization overhead and therefore represents an optimistic upper bound on achievable performance, but it does not preserve FMI step semantics. The inspiration from generic IPC mechanisms is limited to the single-buffer shared-memory producer–consumer data path. No FMI-aware step guard, consumer acknowledgment, or multi-version visibility control is applied in B2-r. Therefore, B2-r is used as a raw-performance upper-bound baseline rather than as a correctness-preserving co-simulation solution.

B2-g (Guarded Single-Buffer SHM) augments the single-buffer design with a step-stamp check at the consumer side. Consumers read the buffer only when the stored step index matches the expected simulation step; otherwise they wait until the correct data become available. While this mechanism partially restores step consistency, it introduces additional synchronization overhead and may lead to straggler effects in broadcast-oriented workloads. Compared with B2-r, B2-g modifies only the consumer admission rule by requiring the stored step stamp to match the expected FMI simulation step. The underlying data layout and single-buffer overwrite policy are otherwise kept unchanged. This design allows us to isolate the effect of minimal step guarding from the effect of double buffering and step-indexed acknowledgments used in the proposed method.

Table 4 summarizes the key design characteristics of the author-implemented communication schemes, including buffering strategy, step protection mechanism, overwrite risk, and compatibility with FMI step semantics. These differences provide the basis for interpreting the performance and correctness trade-offs observed in the experimental evaluation.

4.4. Broadcast-oriented co-simulation scenario

To reflect realistic FMI-based digital twin workloads, we evaluate all communication schemes under a broadcast-oriented co-simulation scenario. Rather than restricting the evaluation to a one-to-one FMU coupling, we consider a one-to-many (1:N) data distribution model, in which a single producer FMU generates environment or sensor data that must be consumed by multiple FMUs within the same simulation step.

This communication pattern is common in maritime and cyber–physical systems. Examples include an environment FMU broadcasting navigation states to multiple ship FMUs, or a camera FMU distributing large image frames to perception, localization, and control FMUs. In such scenarios, the same data must be delivered to many consumers at identical simulation times.

Table 4
Comparison of evaluated communication schemes and their step-semantics properties.

Scheme	Buffering	Step guard	Overwrite risk	FMI-Step
B1: FMI API	None	Master-enforced	None	Yes
B2-r: Single SHM (relaxed)	Single	None	High	No
B2-g: Single SHM (guarded)	Single	Step-stamp check	Low	Partial
P: (proposed) Dual-path	Dual	Step-indexed ack	None	Yes

Under these conditions, conventional FMI API-based communication incurs repeated data transfers and master-mediated synchronization for each consumer, resulting in amplified latency and scheduling overhead as the number of consumers increases. Similarly, relaxed shared-memory designs that lack step-level coordination are prone to temporal inconsistencies when multiple FMUs access shared data asynchronously.

For these reasons, all evaluated schemes are tested primarily under 1:N broadcast configurations, with the number of consumers varied to assess scalability. The one-to-one (1:1) case is included only as a sanity check and does not represent the primary target workload of this study.

The focus on 1:N broadcast is intentional because it stresses fan-out communication and exposes the interaction between large-payload transfer and per-consumer step-consistent visibility. However, this topology does not cover all possible FMI-based interaction patterns. Many-to-many communication may require additional buffer ownership rules, acknowledgment aggregation, and scheduling policies. The present evaluation therefore supports claims for broadcast-oriented workloads, while many-to-many topologies are left for future investigation.

5. Performance evaluation

5.1. Evaluation overview

This section evaluates the performance characteristics of the proposed dual-path FMU communication architecture (P) in comparison with the baseline approaches introduced in Section 4.3. The evaluation focuses on three key aspects relevant to FMI-based co-simulation: temporal performance, scalability under broadcast workloads, and execution correctness.

To separate raw performance limits from FMI-compliant behavior, we consider two variants of single-buffer shared-memory communication. The relaxed variant (B2-r) represents an optimistic upper bound without enforcing step semantics, whereas the guarded variant (B2-g) enforces step-level consistency via step-stamp checks at the cost of additional synchronization.

Unless otherwise stated, clean-condition experiments are conducted at an update frequency of 100 Hz. Each plotted value represents the mean over repeated runs (50 trials with 1000 steps each). Stress-condition experiments introduce application-level sleep-based asynchronous consumer delays to evaluate sensitivity to single-host consumer concurrency and service-time variation.

All performance figures were regenerated using fixed logical x-axis positions to ensure that data points corresponding to the same experimental condition are visually aligned. Payload-dependent plots use categorical x-axis positions corresponding to 64 KB, 256 KB, 1 MB, and 4 MB, while consumer–scaling plots use exact positions corresponding to $N = 1, 2, 4, 8, 16$. Different communication schemes are distinguished by marker shapes and line styles rather than by horizontal displacement.

To improve readability in print, the same visual encoding is used consistently across all figures. B1 is shown as a blue solid line with circle markers, B2-r as an orange dashed line with square markers, B2-g as a green dash-dot line with triangle markers, and P as a red dotted line with diamond markers. Axis-label, tick-label, and legend font sizes were also increased in the revised figures.

5.2. Step latency analysis

Figs. 2 and 3 present the median and tail latency of simulation steps as a function of payload size. Across all schemes, latency increases with payload size due to memory transfer costs, cache pressure, and synchronization overhead.

The FMI API baseline (B1) shows steadily increasing latency as payload grows, reflecting repeated data copies and master-mediated coordination overhead. The proposed scheme (P) closely tracks B1 at small and moderate payload sizes and exhibits only modest additional overhead for the largest payloads due to step-aware synchronization. These results indicate that step latency in the 1:1 configuration is primarily dominated by memory transfer costs associated with payload movement.

The p99 latency results reveal the worst-case responsiveness of each communication scheme. While the overall trend is similar to the median latency, several configurations exhibit elevated p99 values due to transient system effects. These outliers are primarily attributed to process-level OS scheduling interference, cache and TLB pressure during large payload transfers, and occasional synchronization stalls on the experimental platform summarized in Table 2. Despite these disturbances, the proposed architecture maintains bounded tail latency across all evaluated payload sizes, indicating stable step execution even under adverse runtime conditions.

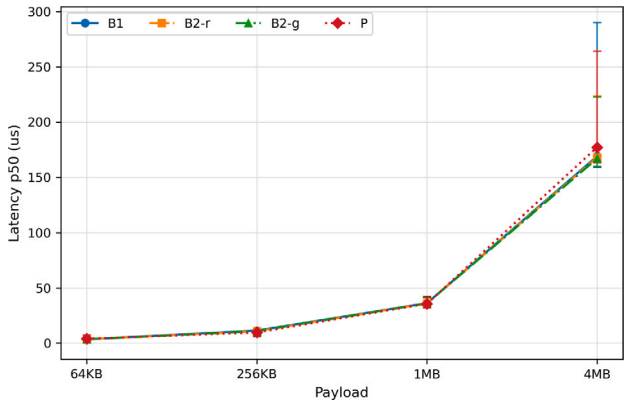


Fig. 2. Step latency (p50) versus payload size under clean conditions at 100 Hz in a 1:1 producer–consumer configuration.

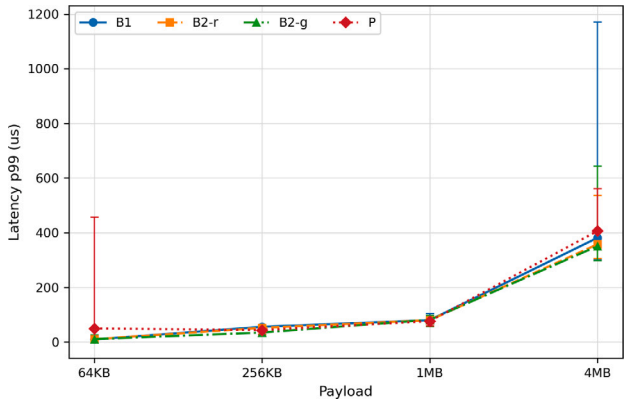


Fig. 3. Step latency (p99) versus payload size under clean conditions at 100 Hz in a 1:1 producer–consumer configuration.

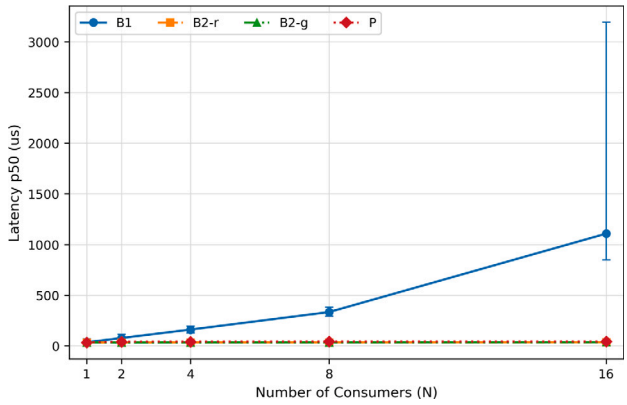


Fig. 4. Median step latency versus number of consumers under a 1:N broadcast configuration in clean conditions. The payload size is fixed at 1 MB and the update frequency is 100 Hz. All schemes are plotted at the same logical consumer-count positions, and error bars indicate run-to-run variability.

To evaluate scalability under broadcast workloads, Fig. 4 shows how the median step latency changes as the number of consumer FMUs increases in a 1:N configuration. A clear trend is observed for the FMI API baseline (B1), whose latency rises rapidly with the fan-out and exhibits substantially increased variability at larger N. This behavior is consistent with repeated data copying and master-mediated coordination overhead, which accumulate as the same payload must be delivered to more consumers.

In contrast, the shared-memory-based schemes (B2-r, B2-g, and P) remain comparatively stable as the number of consumers increases. This indicates that, under clean conditions, the dominant scaling bottleneck in the 1:N broadcast configuration is not

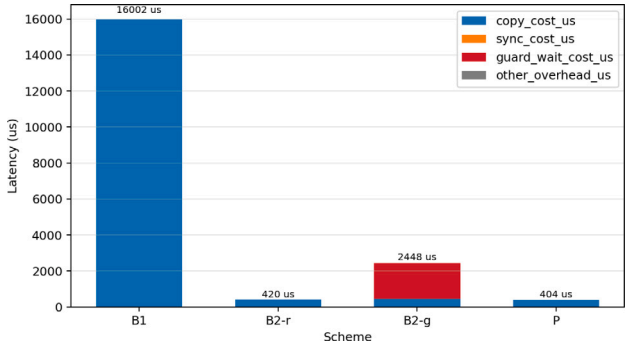


Fig. 5. Step-latency breakdown for a representative stress configuration ($N = 16$, payload = 4 MB, 100 Hz).

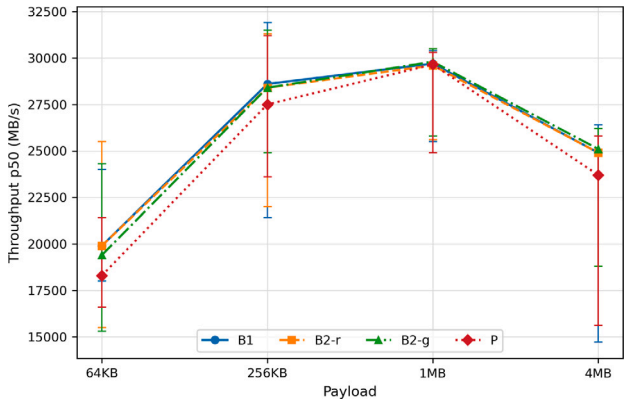


Fig. 6. Effective throughput (p50) versus payload size under clean conditions at 100 Hz.

raw memory sharing itself, but the centralized communication path used by the FMI API baseline. The proposed architecture shows scaling behavior comparable to the relaxed and guarded baselines while preserving step-consistent execution semantics. Therefore, Fig. 4 primarily highlights fan-out scalability, whereas semantic correctness differences are evaluated separately in the stress-condition experiments.

While Fig. 4 reveals differences in fan-out scalability, it does not directly explain the source of the observed latency overhead. To better understand these differences, we analyze the internal latency components of each communication scheme. Fig. 5 presents a representative step-latency breakdown for a demanding broadcast configuration ($N = 16$, payload = 4 MB). The bars show how copy operations, synchronization, guard waiting, and residual overhead contribute to the total latency.

The FMI API baseline (B1) is dominated by copy cost due to repeated data transfers across multiple consumers, which leads to the severe fan-out penalty observed in the scaling experiment. The relaxed shared-memory scheme (B2-r) achieves the lowest raw latency by eliminating synchronization and coordination overhead, but it does not guarantee overwrite safety. The guarded single-buffer scheme (B2-g) prevents overwrite hazards but introduces substantial guard-wait overhead caused by implicit coordination delays between producers and consumers. In contrast, the proposed architecture introduces a small explicit synchronization cost while avoiding the large implicit waiting overhead observed in B2-g. As a result, the synchronization overhead remains bounded even under high fan-out conditions while preserving correct communication semantics.

5.3. Throughput analysis

Fig. 6 reports the effective throughput measured at the median (p50) under clean conditions. The relaxed shared-memory baseline (B2-r) achieves the highest throughput because it eliminates synchronization overhead and allows direct memory access. The FMI API baseline (B1) shows lower throughput due to repeated data copies and master-level coordination overhead. The proposed scheme exhibits slightly lower peak throughput than B2-r for small payload sizes because it introduces explicit synchronization to preserve step semantics.

However, raw throughput alone does not fully capture the suitability of a communication mechanism for FMI-based co-simulation. The relaxed baseline achieves higher throughput by sacrificing overwrite protection and step consistency. In contrast, the proposed architecture maintains competitive throughput while preserving deterministic step semantics and communication integrity. Therefore, throughput should be interpreted together with temporal feasibility and semantic correctness rather than as

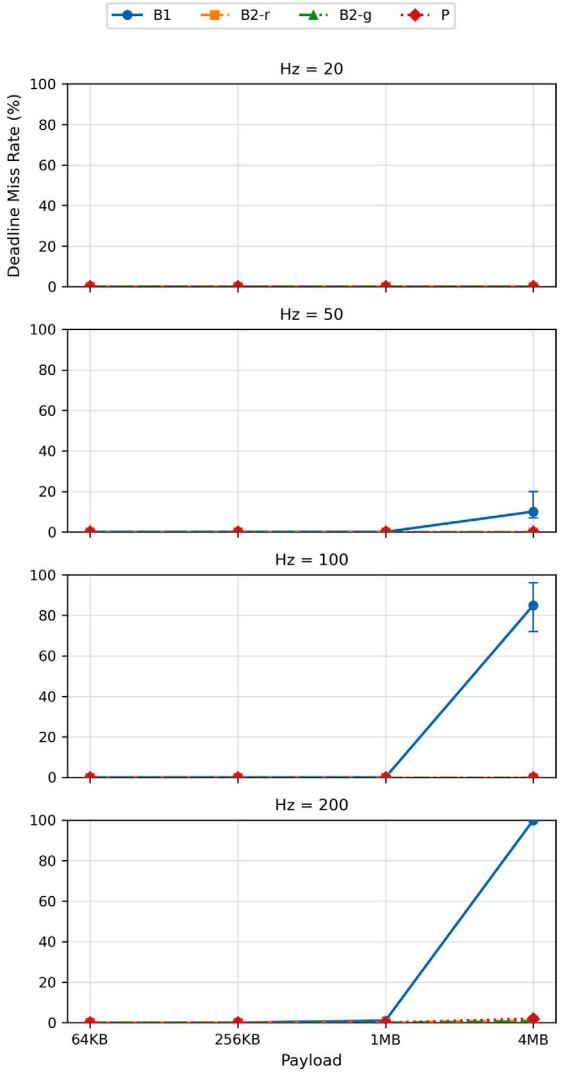


Fig. 7. Deadline miss rate under stress conditions ($N = 16$).

an isolated performance metric. Although relaxed shared-memory schemes can achieve higher raw throughput, they may expose partially updated or overwritten data to consumers when multiple FMUs access the same buffer concurrently. Such inconsistencies violate FMI step semantics and can invalidate simulation results even when temporal deadlines are satisfied.

5.4. Stress conditions and temporal feasibility

Clean-condition experiments reflect ideal execution environments where communication latency is dominated primarily by deterministic data movement costs. In practical co-simulation deployments, however, runtime behavior is often affected by asynchronous execution patterns, consumer-side service-time variation, and shared resource contention within the operating system. To evaluate robustness, we introduce stress conditions that emulate single-host process-level consumer delay. These scenarios amplify straggler effects and reveal differences in synchronization robustness across communication schemes.

The stress results quantify sensitivity to single-host consumer concurrency and sleep-based asynchronous consumer delay rather than to controlled operating-system scheduler perturbations or distributed-network faults. A deadline miss was recorded when the measured producer-side step latency exceeded the nominal period $10^6/f$ microseconds at frequency f, and the deadline-miss rate was computed as the percentage of such producer steps. Because the identical delay distribution was injected into every scheme's consumer processes, differences among the schemes reflect how their synchronization and buffering semantics propagate consumer-side delay under the same configured stress envelope.

Fig. 7 evaluates temporal feasibility by measuring deadline miss rates under stress conditions. A deadline miss occurs when the wall-clock execution time of a simulation step exceeds the nominal simulation period.

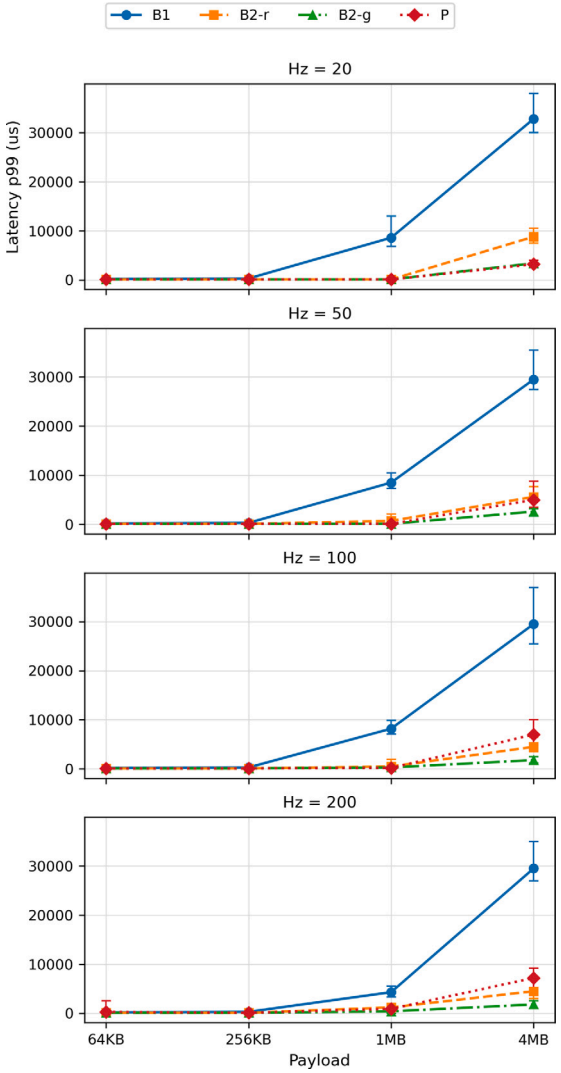


Fig. 8. Tail latency (p99) under stress conditions ($N = 16$).

The API baseline (B1) exhibits increasing miss rates at higher frequencies and payload sizes due to centralized coordination overhead. The relaxed single-buffer schemes maintain near-zero miss rates but do not guarantee semantic correctness, as discussed below.

The proposed architecture maintains low miss rates across most configurations, demonstrating that correctness-preserving synchronization does not necessarily compromise temporal feasibility.

Tail latency provides additional insight into robustness against straggler effects. Fig. 8 therefore reports the p99 latency under stress conditions. The API baseline shows rapidly increasing p99 latency due to amplified straggler effects. In contrast, the proposed scheme maintains lower and more stable tail latency by decoupling data transfer from step synchronization.

5.5. Determinism and correctness under stress

Temporal feasibility alone does not guarantee correct co-simulation execution. Therefore, we evaluate semantic consistency and data integrity.

Fig. 9 reports step-stamp mismatches, which indicate violations of simulation-step consistency, where consumers observe data from inconsistent simulation steps due to overwrite hazards.

Both single-buffer baselines exhibit increasing mismatches as payload size grows, indicating that overwrite hazards can occur even when deadline miss rates remain low. The absence of mismatches in the proposed architecture confirms that the dual-path communication design successfully preserves step-level visibility semantics even under severe scheduling interference.

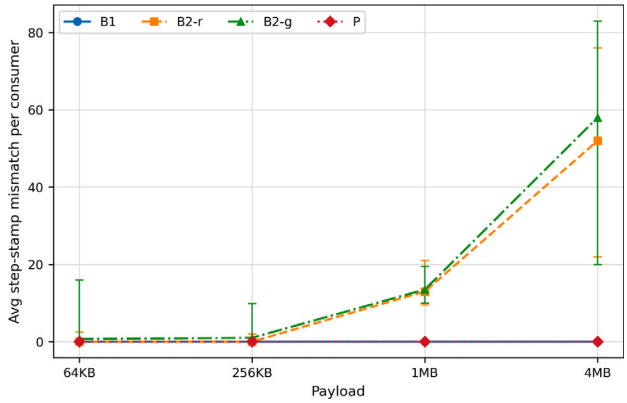


Fig. 9. Step-stamp mismatches under stress conditions at 200 Hz ($N = 16$).

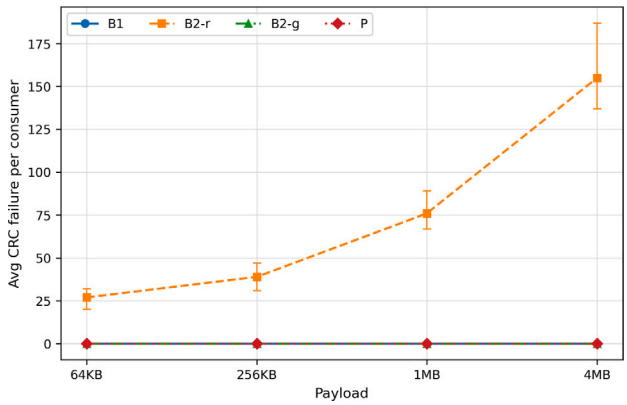


Fig. 10. CRC failures under stress conditions at 200 Hz ($N = 16$).

Fig. 10 reports CRC failures that indicate torn reads or partially written data caused by concurrent access to shared memory buffers. The relaxed baseline exhibits CRC failures at large payload sizes, whereas both the API baseline and the proposed scheme maintain zero failures. The results confirm that the proposed architecture prevents concurrent read–write races while maintaining high communication efficiency.

5.6. Feasible operating region

Fig. 11 summarizes the stress-condition results over the (frequency, payload) parameter space. Each cell is derived from the underlying stress-condition measurements for the corresponding frequency and payload configuration. A configuration is considered feasible only when three criteria are simultaneously satisfied: deadline compliance, step-level semantic consistency, and data integrity verified through CRC validation. The score is therefore not a weighted performance metric; rather, it is a binary feasibility indicator summarizing whether all timing and correctness criteria are met.

The baseline schemes exhibit different failure modes across the parameter space. The FMI API baseline (B1) frequently violates deadline constraints at higher frequencies or larger payload sizes due to repeated data copies and centralized coordination overhead. The relaxed shared-memory scheme (B2-r) maintains high raw throughput but suffers from step-stamp mismatches and CRC failures when concurrent consumers access shared buffers without strict synchronization. The guarded single-buffer scheme (B2-g) prevents overwrite hazards through explicit guard checks, but the resulting guard-wait delays accumulate under broadcast-oriented workloads and can lead to deadline misses.

In contrast, the proposed architecture maintains the largest contiguous feasible operating region. This indicates that the dual-path design can sustain both temporal feasibility and semantic correctness across a broader range of simulation frequencies and payload sizes. The contiguous feasible region further suggests stable operating behavior under stress conditions, which is particularly important for large-scale digital-twin and cyber–physical system simulations where both timing constraints and deterministic data exchange must be preserved. In particular, the fragmented feasibility regions observed in the baseline schemes indicate unstable execution behavior under stress conditions, whereas the proposed architecture maintains a stable and contiguous feasible region. This result highlights that communication-layer design directly determines the practically usable operating region of FMI-based co-simulation systems.

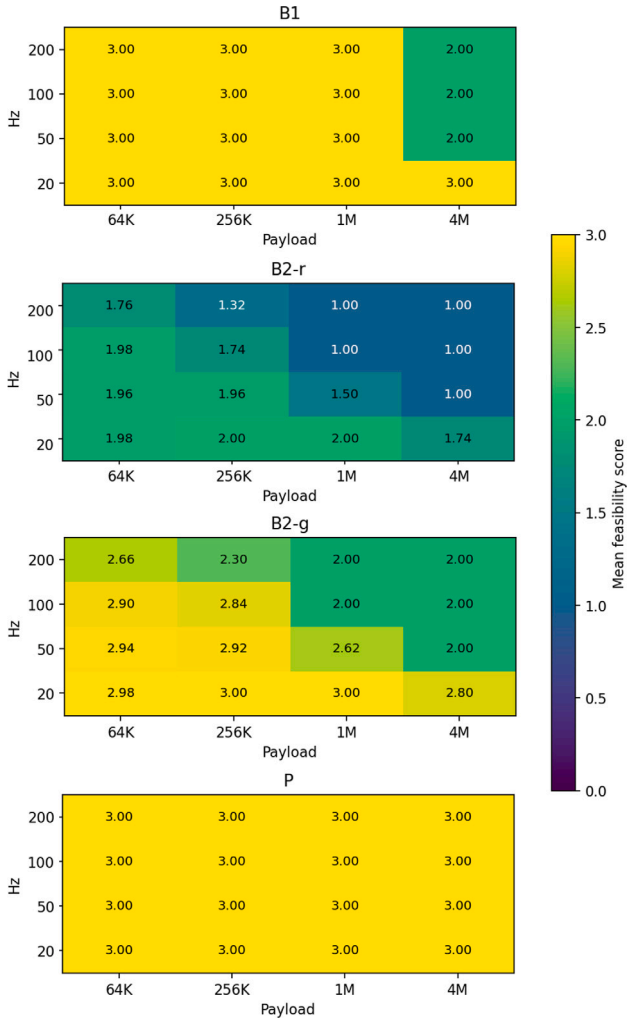


Fig. 11. Feasible operating region under stress conditions ($N = 16$). Each cell is derived from the corresponding stress-condition measurements. A configuration is marked feasible only when deadline compliance, step-stamp consistency, and CRC integrity are simultaneously satisfied. The score represents a binary feasibility indicator rather than a weighted performance metric.

5.7. Summary

The experimental results demonstrate that evaluating FMI-based co-simulation mechanisms requires considering temporal feasibility, scalability, and correctness jointly. Relaxed shared-memory communication achieves high raw throughput but suffers from semantic inconsistencies under stress. The API-based baseline preserves correctness but exhibits scalability limitations at higher frequencies and payload sizes.

The proposed architecture provides a balanced solution by combining efficient shared-memory transfer with step-aware synchronization and integrity protection. As a result, it achieves stable latency scaling, competitive throughput, and correctness-preserving execution across the widest feasible operating region among the evaluated schemes.

6. Discussion

6.1. Temporal performance vs. Simulation correctness

A key observation from the experimental results is that temporal performance alone is insufficient to evaluate communication mechanisms for FMI-based co-simulation. Several communication schemes achieve low latency and high throughput under ideal conditions, yet such metrics do not necessarily reflect whether simulation-time semantics are preserved. Relaxed shared-memory

communication (B2-r) achieves the highest raw throughput and low deadline miss rates in many configurations. However, the stress-condition experiments reveal that such approaches can introduce step-stamp mismatches and CRC failures when multiple consumers concurrently access shared data.

This result highlights an important distinction between *temporal feasibility* and *simulation correctness*. While a system may satisfy wall-clock timing constraints, it can still violate the logical ordering of simulation steps or expose inconsistent data to consumers. In co-simulation environments where multiple models exchange state variables at discrete synchronization points, such inconsistencies can invalidate the overall simulation outcome.

The proposed architecture addresses this issue by explicitly incorporating simulation-step semantics into the communication design. Double buffering preserves the previous step's data until all consumers acknowledge completion, preventing overwrite hazards. As a result, the proposed scheme maintains both temporal feasibility and semantic correctness under broadcast-oriented workloads. Unlike conventional shared-memory communication techniques used in high-performance computing, the proposed architecture explicitly integrates FMI simulation-step semantics into the communication layer. The novelty of the approach therefore lies not in the individual mechanisms such as double buffering or acknowledgments themselves, but in their integration within the FMI master-driven execution workflow to guarantee step-consistent visibility across multiple consumers.

6.2. Scalability in broadcast-oriented co-simulation

Another important finding concerns scalability when a single producer distributes data to multiple consumers. Such broadcast patterns are common in cyber–physical system simulations, digital twins, and distributed sensing scenarios where a shared system state must be propagated to multiple FMUs.

The scaling results in Fig. 4 show that single-buffer designs become increasingly sensitive to straggler effects as the number of consumers grows. When multiple consumers compete for access to a single shared buffer, synchronization overhead increases and slower consumers can delay the visibility of updated data for others. This phenomenon amplifies coordination delays and reduces overall scalability.

The proposed architecture mitigates this limitation by separating bulk data transfer from step-level coordination. Because each simulation step is associated with a dedicated buffer state, consumers can safely read the previous step's data even if the producer has already progressed to the next step. This design reduces contention and enables more stable scaling behavior as the fan-out increases.

6.3. Practical implications for FMI-based co-simulation

The results have several practical implications for the design of FMI-based co-simulation platforms.

First, performance evaluation should not rely solely on latency or throughput metrics. Instead, evaluation must also consider semantic correctness and data integrity, particularly in environments where multiple FMUs interact asynchronously. Although FMI 3.0 introduces binary variables and clock-based execution, migration from established FMI 2.0 ecosystems remains costly in practice. Many industrial co-simulation environments continue to rely on FMI 2.0 due to legacy FMUs, validated toolchains, and certification requirements. The proposed architecture therefore targets improved data-intensive communication while preserving compatibility with existing FMI 2.0 deployments.

Second, communication architectures that appear efficient under ideal conditions may exhibit hidden correctness issues under stress. In particular, overwrite hazards and torn reads can occur when large payloads are exchanged without step-aware synchronization.

Finally, the results suggest that integrating simulation semantics directly into the communication layer can significantly improve robustness without sacrificing scalability. The proposed architecture demonstrates that correctness-preserving communication can remain competitive with relaxed high-performance designs while providing stronger guarantees for deterministic co-simulation execution. While the standard FMI API guarantees semantic correctness through master-mediated data exchange, its centralized communication model introduces repeated memory copies when large payloads must be distributed to multiple consumers. The proposed architecture addresses this limitation by decoupling data transfer from simulation-step coordination, enabling scalable broadcast communication while preserving FMI-compliant execution semantics.

6.4. Limitations and future work

Despite the encouraging results, several limitations remain. The current evaluation focuses primarily on broadcast-oriented workloads with a single producer and multiple consumers. Therefore, the results do not directly cover many-to-many FMU communication patterns or dynamically changing FMU topologies. Such scenarios may involve multiple producers, conflicting write ownership, acknowledgment aggregation, and more complex visibility rules than the 1:N broadcast topology considered here.

An alternative approach to managing concurrent access would be to rely on lock-based synchronization mechanisms such as mutexes or reader–writer locks. However, lock-based designs can introduce blocking delays and priority inversion effects when multiple consumers simultaneously compete for access to shared data. In broadcast-oriented co-simulation workloads, such blocking behavior may amplify straggler effects and degrade temporal predictability. The proposed architecture therefore avoids heavy lock-based coordination and instead separates bulk data transfer from step-level synchronization.

Queue-based or ring-buffer communication mechanisms could also mitigate overwrite hazards by preserving multiple historical states. However, such approaches often require additional flow-control mechanisms or dynamically growing buffers when consumers progress at different rates. While deeper buffering strategies such as multi-slot queues could further reduce coordination delays, they would introduce additional memory management complexity and potentially unbounded buffering requirements. In contrast, the proposed double-buffer design provides a fixed and deterministic memory footprint while preserving step-level visibility semantics required by FMI-based co-simulation. As discussed in Section 3.2, for a payload of size S, the proposed ping–pong structure requires approximately $2S + M$ bytes per shared data object, where M denotes lightweight metadata such as the step index, status flags, CRC value, and acknowledgment state. Thus, the memory usage remains bounded and predictable, but it is approximately twice the payload memory of a single-buffer shared-memory design. This trade-off should be considered when applying the method to embedded or resource-constrained real-time co-simulation platforms.

The shared-memory baselines used in this evaluation are controlled semantic counterparts rather than comprehensive representatives of modern high-performance IPC systems. Advanced mechanisms such as bounded lock-free ring buffers, priority-aware POSIX shared memory, DPDK-style packet processing, or RDMA-based transport may provide different raw performance characteristics. However, such mechanisms do not by themselves define FMI step-indexed visibility, per-consumer acknowledgment, or simulation-time-consistent data validity. Integrating these advanced IPC mechanisms with FMI execution semantics is therefore an important direction for future work.

The present evaluation does not include distributed transport latency, inter-host clock synchronization drift, or partial FMU failures. These factors can affect both timing feasibility and step-consistent visibility in distributed co-simulation settings. Therefore, the experimental claims in this paper are limited to single-host shared-memory execution. Although the present implementation relies on shared memory within a single host, the architectural principle of separating the FMI control path from the high-throughput data path is not inherently restricted to shared-memory transport. Extending the proposed control/data-path separation to distributed middleware such as DDS, RDMA-based transport, or message-oriented communication frameworks requires additional mechanisms for time synchronization, failure detection, and recovery, and remains an important direction for future work.

The paper does not experimentally compare the proposed method with FMI 3.0 binary variables or clock-based execution. Therefore, we do not claim superiority over FMI 3.0 mechanisms. Instead, the proposed architecture should be understood as an FMI 2.0-compatible data-path design that may complement FMI 3.0 execution models by providing step-consistent shared-memory visibility for large payloads. Future work will investigate integration with FMI 3.0 clock-based execution mechanisms and adaptive communication strategies that dynamically select buffering policies based on workload characteristics and runtime conditions.

7. Conclusion

This paper investigated the communication-layer challenges of FMI 2.0-based co-simulation in data-intensive environments. While shared-memory communication can significantly improve raw data-transfer performance, naive designs may violate simulation-step semantics and lead to non-deterministic execution.

To address this problem, we proposed a dual-path communication architecture that separates simulation control semantics from bulk data transfer. The architecture combines shared-memory data exchange with step-indexed synchronization and integrity protection while preserving full compatibility with FMI 2.0 execution semantics.

Experimental evaluation demonstrated that the proposed approach maintains deterministic execution with zero step mismatches and zero CRC failures across all tested configurations. Compared with baseline schemes, the proposed architecture achieves a significantly larger feasible operating region under stress conditions, indicating improved robustness for high-frequency and large-payload co-simulation workloads.

These results suggest that communication-layer design plays a critical role in enabling scalable digital-twin and cyber–physical system simulations under FMI 2.0.

Nevertheless, the current evaluation was performed in a single-node shared-memory environment. Therefore, the results should be interpreted as evidence for single-host FMI 2.0 co-simulation workloads. Extending the proposed architecture to distributed co-simulation environments remains an important direction for future work, particularly in combination with middleware such as DDS, RDMA-based transport, or message-oriented communication frameworks.

Acknowledgments

This research was supported by Korea Institute of Maritime Science & Technology Promotion(KIMST) funded by the Ministry of Oceans and Fisheries, Korea (RS-2022-KS221652, The Development of Simulation-based Evaluation Technology).

Data availability

Data will be made available on request.

References

[1] B.N. Bershad, T.E. Anderson, E.D. Lazowska, H.M. Levy, User-level interprocess communication for shared memory multiprocessors, ACM Trans. Comput. Syst. 9 (2) (1991) 175–198, http://dx.doi.org/10.1145/103720.114701.

[2] C. Scheibe, A. Semerow, J. Menke, P. La Seta, A. Raab, G. Mehlmann, M. Luther, A novel co-simulation concept using interprocess communication in shared memory, in: Proceedings of the 2019 IEEE Power & Energy Society General Meeting, Atlanta, GA, USA, 2019, pp. 1–5, http://dx.doi.org/10.1109/PESGM40551.2019.8973964.

[3] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold, et al., The functional mockup interface for tool independent exchange of simulation models, in: Proceedings of the 8th International Modelica Conference, 2011.

[4] T. Blochwitz, M. Otter, et al., Functional mockup interface 2.0: The standard for tool independent exchange of simulation models, in: Proceedings of the 9th International Modelica Conference, 2012, http://dx.doi.org/10.3384/ECP12076173.

[5] Modelica Association, Functional mock-up interface for model exchange and co-simulation, version 2.0, 2014, Specification.

[6] Modelica Association, Functional mock-up interface specification, version 3.0.2, 2024, Specification.

[7] A. Junghanns, T. Blochwitz, C. Bertsch, T. Sommer, K. Wernersson, A. Pillekeit, I. Zacharias, M. Blesken, P. Mai, K. Schuch, et al., The functional mock-up interface 3.0 – new features enabling new applications, in: Proceedings of the 14th International Modelica Conference, 2021, http://dx.doi.org/10.3384/ecp2118117.

[8] S.T. Hansen, C. Gomes, et al., The FMI 3.0 standard interface for clocked and scheduled execution, Electronics 11 (21) (2022) 3635, http://dx.doi.org/10.3390/electronics11213635.

[9] C. Gomes, C. Thule, D. Broman, P.G. Larsen, H. Vangheluwe, Co-simulation: A survey, ACM Comput. Surv. 51 (3) (2018) http://dx.doi.org/10.1145/3179993.

[10] G. Schweiger, C. Gomes, G. Engel, I. Hafner, J. Schoeggl, A. Posch, T. Nouidui, An empirical survey on co-simulation: Promising standards, challenges and future research needs, Simul. Model. Pr. Theory 95 (2019) 148–163, http://dx.doi.org/10.1016/j.simpat.2019.05.001.

[11] P. Palensky, A.A. van der Meer, C.D. López, A. Joseph, K. Pan, Cosimulation of intelligent power systems: Fundamentals, software architecture, numerics, and coupling, IEEE Ind. Electron. Mag. 11 (1) (2017) 34–50, http://dx.doi.org/10.1109/MIE.2016.2639825.

[12] C. Thule, K.G. Lausdahl, P.G. Larsen, et al., Maestro: The INTO-CPS co-simulation framework, Simul. Model. Pr. Theory 92 (2019) 45–61, http://dx.doi.org/10.1016/j.simpat.2018.12.005.

[13] INTO-CPS Project, Final integration of simulators in the INTO-CPS platform (deliverable D4.3b), 2017, Project Deliverable.

[14] L. Ochel, R. Braun, B. Thiele, A. Asghar, et al., OMSimulator – integrated FMI and TLM-based co-simulation with composite model editing and SSP, in: Proceedings of the 13th International Modelica Conference, 2019, http://dx.doi.org/10.3384/ecp1915769.

[15] T. Thummerer, L. Mikelsons, J. Kircher, NeuralFMU: Towards structural integration of FMUs into neural networks, in: Proceedings of the 14th International Modelica Conference, 2021, http://dx.doi.org/10.3384/ecp21181297.

[16] T. Thummerer, J. Stoljar, L. Mikelsons, NeuralFMU: Presenting a workflow for integrating hybrid neuralODEs into real-world applications, Electronics 11 (19) (2022) 3202, http://dx.doi.org/10.3390/electronics11193202.

[17] L. Lamport, On interprocess communication: part I: basic formalism, Distrib. Comput. 1 (2) (1986) 77–85, http://dx.doi.org/10.1007/BF01786227.

[18] E. Widl, Co-simulation of complex energy systems with MOSAIK and FMI, Automatisierungstechnik (2014) http://dx.doi.org/10.1515/auto-2014-1087.

[19] C. Steinbrink, A.A. van der Meer, M. Cvetković, D. Babazadeh, S. Rohjans, P. Palensky, S. Lehnhoff, Smart grid co-simulation with MOSAIK and HLA: A comparison study, 2018, arXiv:1811.06877.

[20] N. Farrokhseresht, et al., MOSAIK and FMI-based co-simulation applied to transient stability analysis of grid-forming converter modulated wind power plants, Appl. Sci. 11 (5) (2021) 2410, http://dx.doi.org/10.3390/app11052410.

[21] C. Friedrich, A. Lombana, et al., CoFMPy: A python framework for rapid prototyping of FMI-based digital twins, in: ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS-C, IEEE, 2025, pp. 125–128, http://dx.doi.org/10.1109/MODELS-C68889.2025.00027.

[22] L. Willeke, D. Schneider, B. Uekermann, A precice-FMI runner to couple FMUs to PDE-based simulations, in: 15th International Modelica Conference, 2023, http://dx.doi.org/10.3384/ecp204479.

[23] T. Guilbaud, C. Fiorina, S. Lorenzi, A. Scolaro, F. Carminati, D. Maire, A. Pautz, Investigating the functional mock-up interface as a coupling framework for the multi-fidelity analysis of nuclear reactors, Prog. Nucl. Energy 169 (2024) 105022, http://dx.doi.org/10.1016/j.pnucene.2023.105022.

[24] E. Widl, W. Müller, A. Elsheikh, M. Hörtenhuber, P. Palensky, The FMI++ library: A high-level utility package for FMI for model exchange, IEEE Work. Model. Simul. Cyber-Phys. Energy Syst. (MSCPES) (2013) http://dx.doi.org/10.1109/MSCPES.2013.6623316.

[25] C.M. Legaard, D. Tola, T. Schranz, H.D. Macedo, P.G. Larsen, A universal mechanism for implementing functional mock-up units, in: SIMULTECH, 2021, pp. 121–129, http://dx.doi.org/10.5220/0010577601210129.

[26] J. Bastian, C. Clauß, S. Wolf, P. Schneider, Master for co-simulation using FMI, in: Proceedings of the 8th International Modelica Conference, Linköping University Electronic Press, Dresden, Germany, 2011, pp. 115–120, http://dx.doi.org/10.3384/ecp11063115.

[27] S. Park, S. Lee, W. Xu, H. Moon, T. Kim, Libmpk: Software abstraction for intel memory protection keys, in: USENIX Annual Technical Conference, ATC, 2019.

[28] AUTOSAR, AUTOSAR CP SWS crc library, 2024, Specification.

[29] R. Perezaranda, Study of undetected error probability of IEEE 802.3 CRC, 2015, IEEE 802.3bv Task Force contribution.