Reducing HLA Time Management Overhead via Preemptive Execution

Georg Hoelger
DCAITI
Technische Universität Berlin
Berlin, Germany
hoelger@tu-berlin.de

Karl Schrab
Smart Mobility
Fraunhofer FOKUS
Berlin, Germany
karl.schrab@
fokus.fraunhofer.de
OrcID 0000-0002-5083-595X

Robert Protzmann
Smart Mobility
Fraunhofer FOKUS
Berlin, Germany
robert.protzmann@
fokus.fraunhofer.de
OrcID 0000-0002-5531-1936

Ilja Radusch
DCAITI
Technische Universität Berlin
Berlin, Germany
ilja.radusch@dcaiti.com
OrcID 0009-0007-9298-1588

***Abstract*—Distributed co-simulation is widely used to study 5G/6G systems by coupling cellular network simulators with traffic and mobility application models using the IEEE 1516 High-Level Architecture (HLA). When one federate (e.g. a network simulator) generates thousands of fine-grained events per second with zero lookahead, and another (e.g. a traffic simulator) advances in coarse, fixed time steps, the resulting asymmetry incurs severe HLA time-management overhead. In particular when simulating cellular networks, every millisecond a new frame triggers a Next-Event-Request (NER) and Time-Advance-Grant (TAG) exchange with the Runtime Infrastructure (RTI), dramatically slowing overall progress. We propose a conservative, preemptive execution scheme. This simple modification preserves sequential time management, avoids optimistic rollback, and — depending on the network load — reduces wall-clock runtime by 25 – 50 %. The method directly benefits network simulation and modeling, as well as large-scale studies of 5G/6G systems where heterogeneous time scales are unavoidable.**

***Index Terms*—Parallel Discrete-Event Simulation, High-Level Architecture, Federated Simulation, Asymmetric Simulation, Preemptive Time Management, Conservative Synchronization, Time Creep**

I. Introduction

Network simulation is essential for designing and operating modern digital systems. Simulation lets us explore designs quickly and safely without building costly testbeds or disrupting live systems. It scales to large scenarios, is repeatable, and supports controlled what-if experiments, including rare or risky conditions, that are hard or impossible to test in the real world. These strengths matter especially for networks, where tiny timing differences and complex interactions are difficult to observe in field trials. Often, the investigated application performance depends on what happens to packets, frames, and control signals at very small time scales. In connected mobility scenarios, for example, dense cell handovers, radio scheduling, and HARQ behavior directly affect end-to-end delay and reliability for V2X safety messages and cooperative perception. During large events or emergencies, bursts of signaling, paging, and admission control decide who gets service and when, which impacts traffic-management systems and public-safety apps. Multi-access edge computing choices (what to place where, and how to route) depend on radio conditions and backhaul congestion that change quickly over time. Similar cross-layer effects appear in industrial IoT when bursty telemetry competes with control loops that are sensitive to jitter. With accurate modeling of the full network stack, co-simulations deliver realistic QoS estimates, reveal bottlenecks, and support better design decisions, complementing accurate traffic and application models.

Financial support by the Federal Ministry of Education and Research of Germany. Project ID: 16KISK020K.

The study of complex, interconnected systems — such as urban traffic networks coupled with cellular communication infrastructures — often requires federating multiple domain-specific simulators into a single distributed experiment. The IEEE 1516 High-Level Architecture (HLA) has emerged as the de facto standard for this purpose, providing a Runtime Infrastructure (RTI) that ensures consistent time management and data exchange among participating federates [1]. From the network perspective, a key practical challenge is the extreme event rate and fine granularity of cellular stacks: per-frame scheduling ($\approx 1\,\mathrm{ms}$), HARQ, RLC/MAC timers, link adaptation, and control-plane keep-alives produce thousands of short-lived, causally linked events even when user-plane traffic is light. When such an event-heavy network simulator is coupled with traffic or application simulators that advance in coarse time steps (e.g., $100\,\mathrm{ms}$ – $1\,\mathrm{s}$), the heterogeneity in time scales and lookahead creates a synchronization choke point. When simulators exhibit highly asymmetric temporal behaviors HLA's conservative synchronization can become a serious performance bottleneck. Wall-clock time is then dominated not by modeling computations but by cross-simulator synchronization handshakes.

Consider a typical scenario: A microscopic traffic simulator advances in constant one-second time steps, thus enjoying a lookahead of one second, whereas a cellular network simulator inherently offers zero lookahead. In the network domain, link topologies and user-equipment (UE) attachments can change arbitrarily at any moment, precluding any guaranteed minimum message delay. When these two simulators are joined under HLA, every event in the network simulator triggers a Next-Event-Request (NER) from the federate and a corresponding

Time-Advance-Grant (TAG) from the RTI. Even during idle periods, when no user-plane data is transmitted, the network simulator schedules events for connectivity monitoring, and for every frame at 1 ms intervals. The result is a relentless stream of NER/TAG exchanges that dominate CPU usage, a phenomenon that is experienced as "time creep".

We introduce a conservative preemptive execution scheme: upon receiving a TAG for logical time T, the event-heavy federate may proceed up to T without further RTI interaction, given the circumstances that no new external (cross-federate) events do arise before T. Should an external event occur at time at $T' < T$, the federate immediately preempts execution, and signals the new event to the RTI. This mechanism preserves HLA's causality and sequential guarantees while avoiding the complexity of optimistic rollbacks [2], [3]. The experimental results show 25–50 % reduction in wall-clock runtime across a range of network loads. Thus, by allowing one event-heavy federate to execute preemptively within the conservative synchronization, our approach significantly alleviates time-management overheads in asymmetric distributed simulations.

In this introduction, we outline the performance challenge and present a concise overview of our proposed solution. Section II describes the used software architecture. Section III gives an overview of related work and possible solutions to the performance problem. Section IV provides the required theoretical background, and subsequently in section V the new method of preemptive execution is discussed in detail, highlighting both its benefits and limitations. Section VI presents experimental results, and in Section VII we conclude the paper together with a discussion of potential extensions.

II. MOSAIC Architecture

The open-source co-simulation framework Eclipse MOSAIC allows to couple multiple domain-specific simulators [4]. All simulators are coupled through an HLA-inspired interface, so the simulators communicate with each other through a set of predefined so-called interactions. At the core of MOSAIC is the Run-Time Infrastructure (RTI) that ensures the distribution of the interactions to the simulators and also handles the time management of the simulators. The coupling of MOSAIC to the networking simulators (ns-3 and OmNET++) is implemented through a custom socket-based protocol.

In this work, we couple Eclipse SUMO, ns-3, and the MOSAIC Application simulator (see Figure 1). While the vehicle movements are modeled in SUMO, the MOSAIC Application simulator is used to create the network traffic load. By using MOSAIC applications, we can leverage previous work and use already existing applications to build up a realistic load, as well as the ability to map applications to vehicles.

In the current setup of the ns-3 federate, we utilized the ns-3 LENA model to integrate an LTE module for each mobile node [5]. Here we created one Evolved Packet Core (EPC), comming with Packet-Gateway (PGW), the Service-Gateway

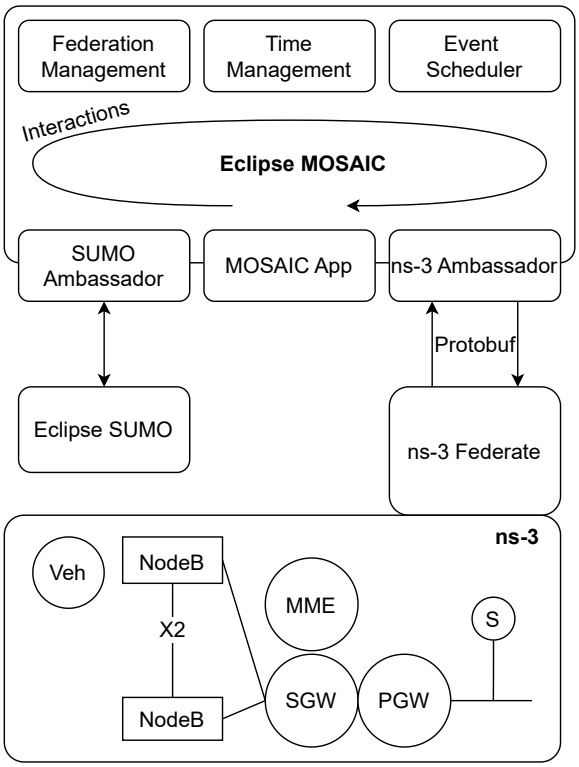

Fig. 1. Coupling of SUMO and ns-3 using the MOSAIC framework.

(SGW), and the Mobile Management Entity (MME). For each configured base station, a corresponding eNB is created along with an X2-Interface, such that eNBs can directly communicate with each other. The utilized ns-3 LENA model mirrors the real network as precise as possible: UEs and eNBs implement PDCP, RLC, MAC, and PHY; an EPC with MME/SGW/PGW provides GTP-U tunneling and connects to ns-3's IP/application stacks. LTE's $1\,\mathrm{ms}$ TTIs trigger frequent scheduling, resource allocation, link adaptation, and HARQ. Each transport block yields error checks, ACK/NACK, and possible retransmissions. RLC timers, segmentation, and PDCP sequencing/ciphering add per-packet work. The core encapsulates/decapsulates in GTP-U, routes by TEID, and handles bearer procedures. All of this translates into a large number of scheduled events. The asymmetry in number of events can also be directly measured. In Table I the amount of events from MOSAIC perspective are shown. Clearly, ns-3 dominates the amounts of events by outrageous dimensions, which is both faithful to the system's operation and a primary driver of computational cost.

TABLE I
Number of Events Triggered in MOSAIC's RTI

Network Load	SUMO	Application	ns-3
Light (50 pkt/s)	166	7078	381 940
Heavy (200 pkt/s)	166	27 500	891 330

III. Related Work

The researches of parallel and distributed simulation focus on how to achieve high-performance simulation while ensuring all events to be parallelly processed and still maintaining their causal relationships [6].

Jafer, Liu and Wainer give an extensive overview about synchronization methods [7] but most of the proposed improvements regard the synchronization problem in a decentralized way, not having the HLA Time Management as central entity.

The Smart Time Management from Huang et al propose a unified HLA time management service likewise for time-stepped, event-driven and optimistic simulators [6]. Sadly, how they point out, this has the direct disadvantage of a performance decrease.

Crues and Dexter discuss HLA Time Management as the challenge to combine several federates with different execution rates [8]. Though, their goal is to have a solid real-time simulation, and they do not even pursue the goal to run the simulation as-fast-as-possible.

Jun Lee at al propose optimizations for a central time management inspired by Lingua Franca [9]. They take advantage of federates that act purely as sensors and thereby only have an outgoing information flow, and secondly that the result of such sensors is only sparsely required. Think of an alarm sensor where an "all good" message does not require any action at the other federates.

Jefferson and Barnes [10] mention preemptive event handlers but only in the context of optimistic synchronization. Here the execution can be preempted in case that a straggler message ("message in the past") arrives in the meantime. Whereas in our case we use preemption to stop execution early enough, so that no straggler message ever occurs.

Optimistic synchronization [2] is ill-suited to the HLA federation setup considered here. Firstly, an optimistic synchronization interface might not be available, and writing the required framework from scratch is a cumbersome and tidious work. Secondly, the typical amount of cross-federate events in our scenario produce a high straggler rate, forcing frequent rollbacks, anti-messages, and continuous state checkpointing. Saving and restoring detailed ns-3 protocol stacks at sub-millisecond granularity is costly in memory, serialization, and cache pressure; the overhead often exceeds any speculative gains. Even if optimistic synchronization is available and in itself beneficial, interoperability with conservative federates is not always given, as Strassburger points out [11].

In summary, there is very little literature, that explores variations and improvements on top of HLAs conservative synchronization, in order to optimize the performance but still omit the complexity of optimistic synchronization.

IV. HLA Time Management [1]

In distributed simulation under the High Level Architecture (HLA), conservative time management is the mechanism by which the Run-Time Infrastructure (RTI) ensures that all events are processed in non-decreasing order and that no federate ever receives an event dated earlier than its own current logical time. The RTI classifies incoming messages as either receive-ordered (RO) or time-stamp-ordered (TSO). RO messages are delivered immediately and in arbitrary order, whereas TSO messages are buffered until the RTI can compute a lower bound on the time stamps of all future messages (the LBTS) and thus guarantee that no smaller-stamped event will later arrive.

Time-regulating federates drive the Lower Bound on Time Stamp (LBTS) computation by declaring the smallest timestamp they may yet generate, while time-constrained federates depend on LBTS to know how far they may advance without risk of receiving past messages. Federates that clear these two flags bypass timestamp ordering altogether and exchange only RO messages. In the remainder of this paper, we assume that we look at time-regulating as well as time-constrained federates, which are both allowed to issue TSO messages and are able to process TSO messages.

Federates cannot advance their logical clocks autonomously; instead, they invoke explicit services and await a callback from the RTI. In a Time Advance Request (TAR), a time-stepped federate requests advancement of its logical time to a specified value T. The RTI then delivers all RO messages and every TSO message with time stamp $\leq T$, and when it can be sure that no further TSO messages stamped less than or equal to T will arrive, it issues a Time Advance Grant (TAG) with parameter T. The TAG callback informs the federate that its logical time has been advanced to T and that no pending TSO messages with smaller stamps remain.

An event driven federate will invoke a Next Event Request (NER) when it has completed all simulation activity at the current logical time, and is ready to advance to a new time. The parameter T specified in the NER request indicates the logical time to which the federate would like to advance, typically, T is the time stamp of the next event in the federate's local set of pending events. If no TSO message with stamp $< T$ exists or will arrive, the RTI issues TAG(T), otherwise it delivers the smallest-stamp event with time T' and issues TAG(T') stamped at that event time.

To accommodate federates requiring zero lookahead HLA provides variants of TAR and NER called Time Advance Request Available (TARA) and Next Event Request Available (NERA). When the RTI issues TAG in response to TARA or NERA, it no longer guarantees that all simultaneous events stamped exactly at T have been delivered; it only guarantees that no events with time stamp less than T are still outstanding. Also when receiving a TAG(T) as response to TARA/NERA, the federate still is allowed to issue messages stamped at time T. This would not be allowed when receiving a TAG(T) as response to TAR/NER.

All the afore mentioned mechanisms and services performed by the RTI constitute the core of HLA time management. They allow federates with diverse internal time-flow mechanisms (time-stepped, event-driven or real-time) to interoperate without revealing their local time-management policies, while preserving causality and repeatability across federation executions.

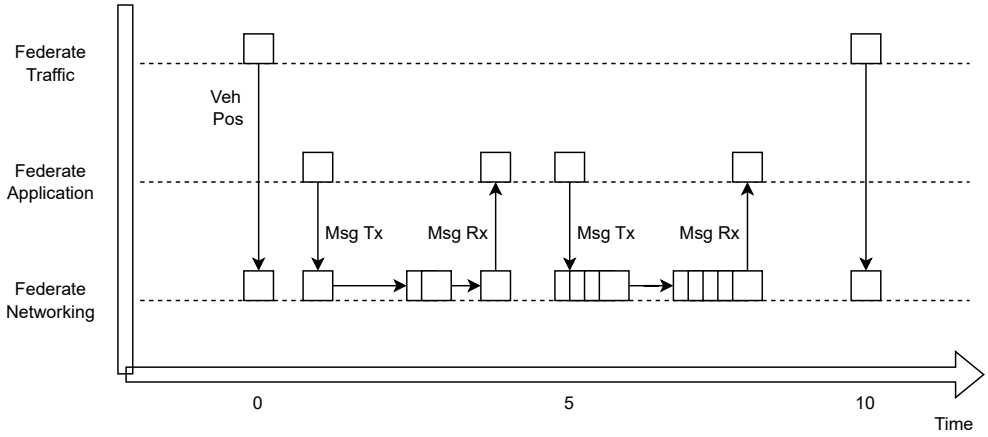

Fig. 2. Typical sequence of events for a simulation incorporating mobility and communication simulation.

V. Our Approach

In Figure 2 you can see a typical sequence of events (arrows indicate causality): The traffic federate is time-stepped, and (in the example of this paper) issues position updates every ten seconds. The application federate generates the communication traffic, which is sent to the network federate. Here the whole network stack is simulated, and a lot of individual events are created, often scheduled very spontaneously; think of a transmission routine which schedules the reception spontaneously just some microseconds later. Eventually, the packet reception is signaled from the network federate to the application federate. Given this scenario, we see, that the network federate will spend a lot of time synchronizing with the RTI Time Management: Every event will need to be requested and granted by the RTI, to guarantee that all federates run in sync.

All three federates (Traffic / Application / Network) are waiting to be allowed to proceed. So the RTI usually has three pending requests. Subsequently we write request 'R' for any TAR/NER/TARA/NERA and grant 'G' for TAG. The RTI will then issue a grant for the earliest request (e.g. $G_T(0)$) and wait for the completion, which is indicated by the request (e.g. $R_T(10)$). Also, the advancing of the traffic federate created an event for the network federate $R_N(0)$. Subsequently, most of the RTIs communication will have the pattern $G_N(now) \rightarrow R_N(next)$.

In our novel approach, the network federate, being the federate with the most of the events by far, will gain a preferred role in the upcoming algorithm. The networking federate has to distinguish between internal events and external events. Internal events are usual events that also happen when the federate runs on its own. External events are cross-federate events, where other federates are involved, so when messages to other federates are sent. Previously the network federate did request any sort of event, no matter if internal or external. In our approach the network federate only requests external events. Pay attention to the fact that the time of the next external event is usually not known to the network federate. In this case, instead of requesting the next internal event time, the network federate will request the next event with empty time: $G_N(now) \rightarrow R_N(\_)$.

Given this change, the RTI has to always grant the same times to the network federate as to the other federates. For any $G_X(T)$ grant the RTI will issue a $G_N(T)$ grant just before.

This mechanism of "permanent upstream execution" guarantees that all external events from the network federate that might not have been announced yet, will be uncovered, before any other federate proceeds to that time. By this we circumvent the problem of other federates receiving a straggler message from the network federate.

Let's say we have an (unknown) message reception at time 4, and next event request from the traffic federate for time 10. Before executing $G_T(10)$ the RTI will first issue a $G_N(10)$ which will generate both the reception $R_A(4)$ as well as the generic request $R_N(\_)$. So although we gave the permission to advance until time 10, the network federate detected an external event at time 4. The event queue at the RTI gets updated and the network federate waits to resume its execution, starting at time 4. Because the network federate individually decides until which time to simulate we call this scheme "preemptive execution".

VI. Evaluation

In order to evaluate the described changes we use Eclipse MOSAIC [4]. For the evaluation we use three coupled simulators:

- SUMO (traffic simulator): To generate realistic moving patterns. Position updates are issued every one second.
- MOSAIC Application (application simulator): To generate network traffic load.
- ns-3 (network simulator): To simulate the communication stack in full detail.

The scenario is kept very simple; there is one car with one application that generates network traffic. The traffic is sent over the LTE interface to a server in the backbone. The simulation itself has 40 seconds of simulation time, the traffic generator is only running from $10\,\mathrm{s}$ to $40\,\mathrm{s}$.

We varied simulation runs with different network traffic loads, from 50 packets per second to 250 packets per second. Each packet has 10 kB in size. Each parameter combination is run five times with different seeds to build an average. As performance indications we look at the overall wall-clock runtime which is tracked by MOSAIC. Additionally we implemented counters in the ns-3 federate, to keep track of TAG/NER calls.

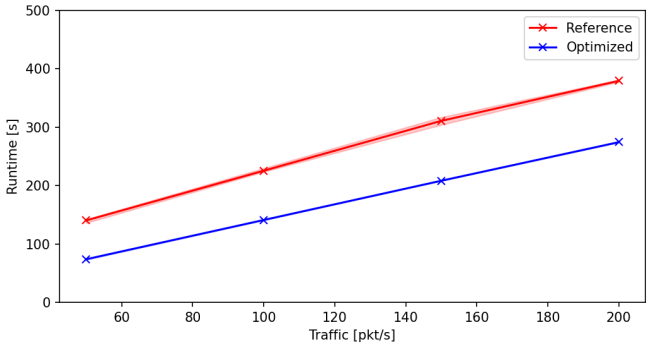

Fig. 3. Wall clock time decreases significantly when using our approach.

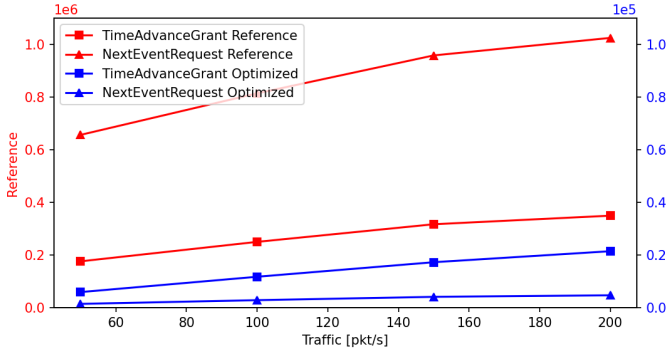

Fig. 4. Count of TAG & NER Events. Note that the number of events in the optimized approach is one order of magnitude lower.

In Figure 3 we can see the simulation runtime for both the reference and optimized run. For the scenario with low network traffic load, the runtime improves from 140 s to 74 s. For the high traffic load scenario the runtime improves from 379 s to 274 s. Thereby the improvement lies between 27 % and 47 %.

Apart from the average runtime, also minimum and maximum runtime are shown in this figure, thereby the red line seems slightly thicker, whereas minimum and maximum runtime are not visible for the blue line. Apparently, reducing the amount of events does reduce the amount of context switches and thereby the deviation of the runtime further decreases.

Figure 4 shows the event counters for both TAG and NER events, counted by the ns-3 federate. It is important to mention that the reference and optimized run is drawn on two different y-axis, which have scales that are different by one order of magnitude. We see that with the optimized protocol, the simulation can be executed using (much) less than factor 10 TAG and NER events.

Also it is interesting to highlight one detail: In the reference version there are more NERs than TAGs, but in the optimized version this flips and there are more TAGs issued, than NERs. To fully understand this, it is important to know some implementation details, because from the pure theory you would assume that the counters for TAG and NER events should be always equal: Firstly, in our implementation a NER is not a direct result of a TAG, ns-3 does issue a NER(T) for any newly scheduled event, telling the RTI that it has to be granted to proceed to T at a later point in time. If a later event is scheduled at the same time T, again a NER(T) will be issued, but only one TAG grant will be necessary to proceed. Secondly, the empty NER(_) calls in our setup are not explicitly signaled but rather implicitly assumed.

In Table II the improvement for both runtime as well as TAG and NER counters are displayed in percentage.

TABLE II
IMPROVEMENT FROM REFERENCE TO OPTIMIZED

Traffic (pkt/s)	Runtime (%)	TAG (%)	NER (%)
50	47.42	96.64	99.78
100	37.45	95.30	99.65
150	33.01	94.55	99.57
200	27.68	93.86	99.54

VII. CONCLUSIONS AND FUTURE WORK

In this paper, we have addressed the challenge of HLA time-management overhead, where zero-lookahead, event-heavy federates introduce unnecessary synchronization to the conservative synchronization scheme, given the asymmetric distribution of events. By introducing a preemptive execution scheme, we enable the event-heavy federate to proceed speculatively up to its granted timestamp without repeated RTI interactions, only preempting execution when a genuine cross-federate event emerges. This easy-to-implement design principle preserves HLA's causality guarantees while dramatically reducing the number of Next-Event-Request/Time-Advance-Grant exchanges, and avoiding the complexity of rollbacks in optimistic synchronization.

Our prototype implementation within MOSAIC, coupling SUMO and ns-3, demonstrates a 25-50 % reduction in wall-clock runtime, caused by an order-of-magnitude decrease in synchronization messages, confirming the efficacy of preemptive execution in realistic cellular-traffic co-simulations. Despite these gains, our implemented solution currently remains for a single event-heavy federate, indicating the need for a more generic framework where any federate can request the privilege of preemptive execution. Looking ahead, we plan to extend our evaluation to large-scale scenarios, and verify under which circumstances more than one federate can make use of the advantage that comes with preemptive execution.

As we build up on open-source projects, we contribute our code for further investigations [12] [13].

ACKNOWLEDGEMENT

The authors acknowledge the financial support by the Federal Ministry of Education and Research of Germany in the programme of “Souverän. Digital. Vernetzt.” Joint project 6G-RIC, project identification numbers: 16KISK020K.

REFERENCES

[1] R. M. Fujimoto and R. M. Weatherly, “Hla time management and dis,” in *Proceedings of 14th Workshop on Distributed Interactive Simulation*, 1996.

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

[3] N. Naumann, B. Schunemann, I. Radusch, and C. Meinel, “Improving v2x simulation performance with optimistic synchronization,” 2009. [Online]. Available: https://publica.fraunhofer.de/handle/publica/365832

[4] K. Schrab, M. Neubauer, R. Protzmann, I. Radusch, S. Manganiaris, P. Lytrivis, and A. J. Amditis, “Modeling an its management solution for mixed highway traffic with eclipse mosaic,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 6, pp. 6575–6585, 2023.

[5] N. Baldo, “The ns-3 lte module by the lena project,” Center Tecnologic de Telecomunicacions de Catalunya, 2011. [Online]. Available: https://cttc-lena.gitlab.io/nr/html

[6] J.-y. Huang, M.-C. Tung, K. M. Wang, and M.-C. Lee, “Smart time management—the unified time synchronization interface for the distributed simulation,” *Computer Standards & Interfaces*, vol. 27, no. 2, pp. 149–161, 2005.

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

[8] E. Z. Crues and D. E. Dexter, “A discussion of time management concepts and time constraint equations for multi-rate federation executions,” in *2025 Simulation Innovation Workshop (SIW)*, 2025.

[9] B. Jun, E. A. Lee, M. Lohstroh, and H. Kim, “Efficient coordination for distributed discrete-event systems,” in *2024 22nd ACM-IEEE International Symposium on Formal Methods and Models for System Design (MEMOCODE)*. IEEE, 2024, pp. 114–118.

[10] D. R. Jefferson and P. Barnes Jr, “Virtual time iii, part 1: Unified virtual time synchronization for parallel discrete event simulation,” *ACM Transactions on Modeling and Computer Simulation*, vol. 32, no. 4, pp. 1–29, 2023.

[11] S. Straßburger, *Optimistic synchronization in the HLA 1516.1-2010: interoperably challenged*. Universitätsbibliothek Ilmenau, 2015.

[12] K. Schrab and G. Hoelger, “Eclipse mosaic,” Computer software, 2025, accessed: 2025-09-22. [Online]. Available: https://github.com/hoelger/mosaic/tree/ICOIN-hla-time-management-preemptive-execution

[13] G. Hoelger, “ns-3 federate,” Computer software, 2025, accessed: 2025-09-22. [Online]. Available: https://github.com/hoelger/ns3-federate/tree/ICOIN-hla-time-management-preemptive-execution