AP-LET: Enabling Deterministic Pub/Sub Communication in AUTOSAR Adaptive

Davide Bellassai[1,2], Claudio Scordino[2], Daniel Casini[1], and Alessandro Biondi[1]

[1]Scuola Superiore Sant'Anna, Pisa, Italy
[2]Evidence S.r.l., Pisa, Italy

***Abstract*—The automotive software industry is facing a paradigm shift driven by the need to develop more and more advanced functionality distributed on multiple electronic control units. The AUTOSAR Adaptive standard has been designed as a service-oriented architecture on top of a general-purpose operating system to tackle this paradigm shift.**

Nevertheless, it does not provide means to ensure deterministic communication, as required in safety-related components.

This paper studies the integration of the System-Level Logical Execution Time (SL-LET) paradigm in AUTOSAR Adaptive.

The key design challenges and requirements to support SL-LET in AUTOSAR Adaptive are described, highlighting how to overcome the considerable differences between the AUTOSAR Classic and Adaptive domains. Then, a meta-protocol named AP-LET is presented, together with two concrete instances: one based on high-priority tasks and another leveraging timestamps in the message payload to handle communications and ensure determinism. A complete implementation of both protocols is also described. AP-LET was finally evaluated with a realistic automotive application, showing its feasibility and effectiveness.

I. Introduction

The automotive industry has historically been rather conservative, relying on simple Electronic Control Units (ECUs) running domain-specific real-time operating systems (e.g., OSEK [1]) and networks (e.g., CAN, LIN, FlexRay) to handle safety-critical functions. This legacy paradigm is matched by the well-established AUTOSAR Classic standard [2]. However, the recent increase in the complexity of automotive systems — due to the integration of new functionalities, such as assisted or autonomous driving — has forced the industry to replace the original signal-oriented design with a modern *service-oriented architecture* (SoA). Using this different paradigm, the various software components are decoupled from each other and communicate by requesting and providing *services*. Each component can thus be designed in isolation, lowering the complexity of the designed system, with benefits also in terms of scalability and reusability. In addition, traditional automotive networks are being replaced with general-purpose networks and stacks (namely, Ethernet and TCP/IP) that allow to both reach higher throughput and reduce the amount of cabling inside the vehicle.

Consistently with this paradigm shift, a new standard has been proposed to extend AUTOSAR Classic. In 2017, the AUTOSAR international consortium started pursuing these goals by creating a novel Adaptive Platform (AP) standard [3]. This new standard is based on POSIX [4] operating systems (e.g., Linux) and a set of C++ libraries to support multi-thread applications. Unlike the previous AUTOSAR Classic specification, however, it has not been designed to guarantee the timing constraints of the executed applications by construction.

The challenge. Despite the need to handle much more flexible systems than those managed by AUTOSAR Classic, next-generation automotive applications still require guaranteeing deterministic inter-task communication and bounded end-to-end latency for a set of relevant chains of software components. In this work, we argue that the *Logical Execution Time* (LET) paradigm represents an effective solution to tackle this challenge. LET, although initially proposed in 2001 for synchronous programming languages [5], attracted renewed attention in the last decade by the automotive industry to face the transition towards multi-core systems. LET enforces data exchange between tasks to logically occur at pre-determined time instants, thus enabling communication determinism without relying on the actual scheduling of tasks. Furthermore, it proved to be beneficial to bound the end-to-end delay of cause-effect chains [6]–[8] and reduce memory-contention delays [9], possibly leveraging direct memory access engines for improved communication performance [10].

System-Level Logical Execution Time (SL-LET) [11] was later proposed extending LET by coping with the non-negligible duration of data transfers through networks in distributed systems. Being AUTOSAR Adaptive conceived to handle multi-ECU, distributed systems running dynamic workloads, SL-LET represents a perfect match to enable deterministic communication. Nonetheless, integrating SL-LET within AUTOSAR Adaptive is *nothing but straightforward* due to two major, non-trivial difficulties:

1) SL-LET assumes communication between periodic tasks (activated by timers); instead, in AUTOSAR Adaptive, communication is message-driven and follows a service-oriented architecture (as for the pub/sub paradigm [12]) and works with tasks that are mostly activated by data reception.
2) Traditional LET implementations [9] typically consider static systems; instead, AUTOSAR Adaptive supports tasks that can dynamically join and leave the LET paradigm at runtime. LET, therefore, needs to be rethought to cope with the peculiarities of AUTOSAR Adaptive.

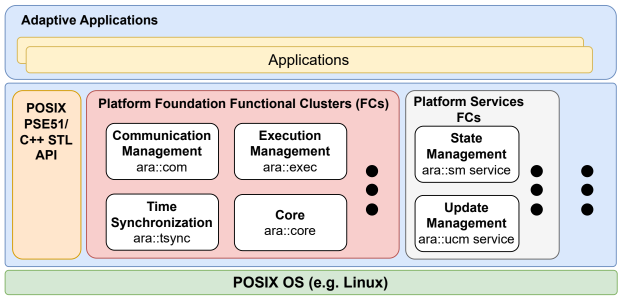
Adaptive Applications
Applications
POSIX PSE51/ C++ STL API
Platform Foundation Functional Clusters (FCs)
Communication Management ara::com
Execution Management ara::exec
Time Synchronization ara::tsync
Core ara::core
Platform Services FCs
State Management ara::sm service
Update Management ara::ucm service
POSIX OS (e.g. Linux)

Fig. 1: Architecture of AUTOSAR Adaptive [3].

Contribution. This paper investigates the integration of SL-LET with the AUTOSAR Adaptive and POSIX standards. It proposes a novel meta-protocol named AP-LET and two concrete specializations of the same, discussing the corresponding implementations and the design principles of the meta-protocol that allow it being compatible with Adaptive. The two approaches are then compared to show advantages and disadvantages, both qualitatively and quantitatively. Implementations of AP-LET are also discussed. A quantitative comparison is presented in the form of an extended experimental evaluation based on the WATERS 2017 Challenge by Bosch [13] and the Brake Assistance Demonstrator of AUTOSAR Adaptive, which are representative of industrial-grade automotive applications.

II. Background

This section provides the key background information on AUTOSAR Adaptive, LET, and SL-LET.

AUTOSAR Adaptive. The Adaptive Platform is a modern service-oriented architecture (SoA) built on top of a POSIX-enabled operating system [4]. Functionalities and services are grouped in a few "Functional Clusters" accessible through a C++ API, called *Runtime Environment for Adaptive Applications (ARA)*. Figure 1 shows the architecture of the Adaptive Platform and the services offered by ARA. Communication Management (`ara::com`) and Execution Management (`ara::exec`) are the clusters in charge of service-oriented communication and application lifecycle, respectively.

The standard does not mandate any specific communication protocol (although SOME/IP [14] is recommended), but a growing interest has persuaded the consortium to include the Data Distribution Service (DDS) protocol inside both the Classic and Adaptive specifications [15]. Message-passing communication between software components (SWCs) is based on the client-server pattern, where servers offer a set of *services* that might be invoked by clients, including: **(i)** *Events*: allow clients to be notified about events happening on the server side; **(ii)** *Methods*: allow servers to expose remote procedure calls (RPC) to be invoked by clients; and **(iii)** *Fields*: provide access to data values that clients can remotely get and set.

The interfaces for these services are described at design time through ARXML files. The binding between clients and servers is then performed at run-time via service discovery.

LET. First introduced as part of the GIOTTO framework [5] in the context of synchronous languages, the LET paradigm is used in the automotive sector for multi-core intra-ECU communications, e.g., in the context of AUTOSAR Classic.

LET enforces determinism of read/write communication between periodic tasks. Without LET, tasks communicate at scheduling-dependent time instants, introducing variable jitters that depend on the interference received from other tasks. This phenomenon also appears under the implicit communication paradigm [16] of AUTOSAR, which mandates reading/writing data at the beginning/end of the task execution only, possibly leveraging local data copies. In contrast, LET allows removing any output jitter by logically enforcing communications at predetermined time instants, such as the beginning and end of the task period. Note that the actual read/write operations are not strictly required to happen at these time instants, but the system should behave as this was the case. Most commonly, the beginning and end of the logical execution for an instance of a periodic task are assumed to coincide with the release times of the considered instance and the following periodic instance, respectively, i.e., communications occur at the task's period boundaries. In this way, communications are time-deterministic, which also implies data-flow determinism[1]. Most commonly, the beginning and end of the logical execution of an instance of a periodic task are assumed to coincide with the release times of two consecutive periodic instances, i.e., communications occur at the task's period boundaries.

Sometimes, LET is implemented through shadow data copies and dedicated high-priority tasks that perform the actual read/write operations. This approach allows for ensuring predictable memory access times, which in turn enables accurate control of memory contention [9].

SL-LET. The original LET model assumed logically instantaneous communication, preventing its adoption to distributed systems. Indeed, two main challenges arise when considering distributed ECUs: **(i)** the clock of different ECUs may not be perfectly synchronized, and **(ii)** the time required to perform communications cannot be neglected. To overcome (i), SL-LET [11, 18] assumes that each ECU has its own local time base and that the difference between any two local time bases is bounded by a synchronization error. More generally, the scope of a local time base is called *time zone*. The concept of time zone is also used to cope with (ii). The communication delay between any two time zones is modeled as an *interconnect task*. This task has the purpose of hiding the varying communication delay that can be incurred while traversing networks with a *logical time*. In this way, messages between time zones become visible only when their logical time expires, even if they arrived earlier. The logical execution time of the interconnect task must be higher than *the sum of the worst-case communication latency between the two time*

[1]A producer task producing data for a consumer task is said to communicate in a data-flow deterministic fashion if each instance of the consumer task always reads from the same instance of the producer task in any execution run [17].

zones plus the synchronization error [11]. Within the same time zone, the original LET model can be applied.

III. System Model

This paper considers a system composed of multiple ECUs interconnected by the network (e.g., based on Ethernet). The symbol $\mathcal{E}$ denotes the set of all ECUs.

Each ECU $E_s \in \mathcal{E}$ comprises a multi-core platform and manages a set of real-time tasks. The set of cores included in E_s is denoted by $\mathcal{C}_s$. The symbol $c_{s,k}$ denotes instead the k-th core within $\mathcal{C}_s$. As required by SL-LET, the clock synchronization error between two ECUs must be bounded. To this end, $\delta_{s,d}$ denotes the maximum clock synchronization error between two arbitrary ECUs $E_s, E_d \in \mathcal{E}$.

We consider a set Γ of real-time tasks, in which the i-th task is denoted as $\tau_i \in \Gamma$. Each task releases a potentially infinite sequence of instances (jobs) and is characterized by a worst-case execution time C_i, a period T_i, and a relative deadline $D_i \le T_i$.

Scheduling. Each ECU runs the AUTOSAR Adaptive middleware on top of a POSIX-compliant Operating System (OS) embedding the PSE51 (Portable Operating System Interface), such as Linux. POSIX PSE51 is one of the four real-time profiles defined by the POSIX standard. POSIX (and hence Linux) provides different schedulers, including the fixed-priority real-time and general-purpose (`SCHED_OTHER`) schedulers. The fixed-priority scheduler comes into two different variants, which provide slightly different behavior for tasks assigned to the same priority: the `SCHED_RR` scheduler, which schedules equal-priority tasks in a round-robin manner with a given time slice, and `SCHED_FIFO`, which releases the processor only on suspension, termination or preemption [19]. All schedulers are preemptive.

This work leverages the `SCHED_FIFO` scheduler, i.e., classical preemptive fixed-priority scheduling, which AUTOSAR Adaptive also uses. We consider the partitioned scheduling paradigm, i.e., each task is mapped to one core and cannot migrate across cores: this is known to be the preferred choice by the automotive market thanks to the lower overheads incurred and the more mature literature in terms of real-time analysis technique [20, 21], which can re-use decades of results for single-processor scheduling. Furthermore, also LET and SL-LET consider partitioned scheduling [9, 17].

The set Γ_s denotes all the tasks allocated to ECU E_s while $\Gamma_{s,k}$ denotes all the tasks allocated to core $c_{s,k}$ of ECU E_s.

Service-oriented communication. Tasks communicate by leveraging a service-oriented architecture, in which they subscribe to and/or provide *services*. Services implement logical communication channels between tasks: a task acting as a service provider is essentially a producer sending messages to all the subscribers (i.e., consumers) interested in its service.

This paradigm is similar to the well-known publish/subscribe communication. However, AUTOSAR Adaptive sits on a higher level to abstract how the SoA is actually implemented. The publish/subscribe paradigm is, hence, a possible implementation option, leveraging SOME/IP [14] or DDS [22].

The set of all services is denoted by Θ. Each service $\theta_j \in \Theta$ has a single producer while it can have multiple consumers. A task can be a producer, providing a service, a consumer, by subscribing to the service, or both. Function $\mathcal{P}(\theta_j)$ returns the task that produces messages about θ_j; instead, $\mathcal{S}(\theta_j)$ returns the set of tasks subscribing to θ_j. Service-oriented communications can also involve tasks executing on different ECUs, as described next (Sec. IV-B). Therefore, we distinguish between **(i)** *intra-ECU* and **(ii)** *inter-ECU* communications.

A communication that involves a service θ_j, a producer $\tau_p = \mathcal{P}(\theta_j)$, and a consumer $\tau_r \in \mathcal{S}(\theta_j)$, is said to be *intra-ECU* if both the tasks are in the same ECU, i.e., $\tau_p \in \Gamma_s \wedge \tau_r \in \Gamma_s$, with $E_s \in \mathcal{E}$. Instead, a communication is said to be *inter-ECU* if the two communicating tasks are allocated to cores in two different ECUs, i.e., $\tau_p \in \Gamma_s \wedge \tau_r \in \Gamma \setminus \Gamma_s$. As required by SL-LET, the *worst-case transmission time* (WCTT) between two ECUs $E_s \in \mathcal{E}$ and $E_d \in \mathcal{E} \setminus E_s$ for sending/receiving the data related to a service $\theta_j \in \Theta$ must be known and bounded. Essentially, the WCTT $\omega^j_{s,d}$ bounds the maximum time needed for performing communication between a producer τ_p in $E_s \in \mathcal{E}$ and a consumer τ_r in $E_d \in \mathcal{E}$ for a service $\theta_j \in \Theta$, which depends on the size of the sent/received data, and includes the time needed for context switches, execution of the network drivers and stack, and the network delay.

Table I summarizes the main symbols used throughout the paper.

Symbol	Description
$\mathcal{E}$	set of all ECUs
E_s	s-th ECU
$\mathcal{C}_s$	set of cores within E_s
$c_{s,k}$	k-th core within $\mathcal{C}_s$
$\delta_{s,d}$	max. synchronization error of E_s and E_d
Γ	set of all real-time tasks
τ_i	i-th real-time task
C_i	worst-case execution time for τ_i
T_i	period of τ_i
D_i	relative deadline of τ_i, $D_i \le T_i$
Γ_s	set of all tasks allocated to E_s
$\Gamma_{s,k}$	set of all tasks allocated to core $c_{s,k}$ in E_s
Γ^k_L	set of tasks registered to AP-LET and executing on core k
Θ	set of all services
θ_j	j-th service
$\mathcal{P}(\theta_j)$	function returning the task producing messages on θ_j
$\mathcal{S}(\theta_j)$	function returning the set of tasks subscribing to θ_j
$\omega^j_{s,d}$	WCTT between ECUs E_s and E_d for service θ_j

TABLE I: Main notation used throughout the paper.

IV. The AP-LET meta-protocol

This section presents AP-LET, an AUTOSAR Adaptive-specific meta-protocol that allows matching AUTOSAR Adaptive with state-of-the-art LET/SL-LET proposals. The meta-protocol is instantiated in two different variants: **(i) HP-AP-LET**: (High-Priority AP-LET) based on high-priority LET tasks on both the producer's and consumer's sides; **(ii) TM-AP-LET**: (TiMestamp AP-LET) it uses a timestamp in the payload of the message to distinguish whether a message is valid at the destination.

Both variants satisfy the following requirements:

RQ1 Communications must be deterministic.

RQ2 The differences in the communication paradigms must be managed. Indeed, Adaptive is based on the service-oriented paradigm, in which consumer tasks are triggered by the reception of messages from the services they subscribe to. Very differently, LET (and SL-LET) is designed to work with subscribers that are triggered by periodic stimuli and not in a message-driven fashion.

RQ3 A dynamic network of producers/consumers must be supported. Indeed, differently from AUTOSAR Classic, Adaptive allows tasks to join and leave AP-LET at runtime. Thus, a dynamic mechanism to register the subscribers of each service is required.

RQ4 Producers need to transparently send messages using the same Application Programming Interface (API), independently of whether the consumers are in the same or in different ECUs, despite the two cases being handled differently under SL-LET (Sec. II).

A. Design Principles

Next, we discuss how to address the design requirements in an actual protocol. **RQ1**, **RQ2** and **RQ3** are protocol-related requirements; therefore, they must be considered when designing AP-LET. Differently, **RQ4** is an implementation-related requirement and is discussed later in Sec. V.

Determinism (RQ1). The first requirement is the most common for a LET protocol: determinism. In LET communications, there are two notions of determinism: time determinism and data-flow determinism. In a system with time-deterministic communications, it is possible to predict with certainty the timing in which data will be sent and received. Data-flow determinism is, instead, a weaker property: a task τ_p producing data for a consumer task τ_c is said to communicate in a data-flow deterministic fashion if each consumer job $\tau_{c,r}$ always reads from the same job $\tau_{p,g}$ [17]. Therefore, time-determinism allows ensuring data-flow determinism. For dynamic systems such as those managed by Adaptive, data-flow determinism is still a mandatory requirement, while time-determinism is an optional (but desirable) requirement.

Message-driven vs. time-driven communications (RQ2). As LET and SL-LET are typically designed to work with periodic tasks, Adaptive's service-oriented communication triggers tasks following a message-driven activation pattern. Different design solutions can be devised to overcome this issue.

We underline that most of the software running on automotive ECUs is periodic rather than message-driven. This paper, therefore, follows the most natural approach for this application domain, privileging periodically triggered tasks. Moreover, since the AUTOSAR Adaptive specifications allow the usage of common C++ functionalities, our design optimizes the performance of *intra-ECU* communications by leveraging shared-memory buffers rather than the message-passing mechanisms offered by the `ara::com` layer.

Inter-ECU communications, instead, are message-driven activities, since they necessarily require the usage of the network layer API (e.g., sockets). ECUs receiving messages leverage *listener* tasks, which have the purpose of receiving the message from the network and copying it into the ECU's local memory. After that, SL-LET communication can be implemented by following two different paradigms, which are detailed next. Listener tasks are also leveraged by communication middlewares, such as the FastDDS implementation of the DDS standard [23], in which, instead, a fully non-LET message-driven communication paradigm is used.

This approach minimizes the number of message-driven communications, with benefits on the overall timing predictability: indeed, it is known [23]–[25] that message-driven tasks' activations introduce scheduling effects that must be accounted for in the timing analysis and typically results in an increased number of interfering job instances (and thus a higher interference) on low-priority tasks. Furthermore, message-driven communication in AUTOSAR Adaptive always leverages the network stack, introducing overheads on the system. By keeping the duration of message-driven activities at the bare minimum (only for the time required to copy the data from the I/O buffers to the ECU memory), the additional interference introduced on lower-priority tasks and the communication overheads are minimized.

As shown later, these design choices led to a time-predictable SL-LET solution. Nevertheless, other solutions are possible — a different paradigm can, for example, leverage message-driven communications and implement SL-LET by programming the communication channels to deliver messages (and thus trigger tasks' activations) only after the logical execution time associated with the specific communication has expired. However, such an approach would lead to a more complex implementation of the buffering mechanism at the channel level and of the corresponding LET-driven expiration mechanism. Nonetheless, this design solution could still be beneficial for intrinsically message-driven systems (e.g., DDS-based IoT and edge systems), which typically do not exhibit periodic activations of real-time tasks. This research direction is left for future work.

Supporting dynamic producers/consumers (RQ3). As AUTOSAR Adaptive allows tasks to join and leave the service-oriented architecture dynamically, AP-LET must provide mechanisms to *register* and *deregister* tasks to services. This constitutes another considerable difference from classical automotive architectures, in which tasks are statically declared to be part of communication at the design time.

B. Definition of AP-LET

This section presents the AP-LET Meta-Protocol. AP-LET is specialized to provide SL-LET communications on the service-oriented communication architecture of AUTOSAR Adaptive. AP-LET manages intra-ECU communications based on classical shared-memory communication [26]; however, its implementation has been deeply revisited to cope with the peculiarities of AUTOSAR Adaptive (see Sec. V). Inter-ECU communications are instead managed in two different ways, giving rise to the two instances of the meta-protocol: **HP-AP-LET**, a variant based on high-priority tasks managing communications and **TM-AP-LET**, a variant that leverages message timestamps to implement SL-LET.

The meta-protocol is defined by the following set of rules.

AP1 Registration phase. When joining AP-LET, tasks must specify their *task id*, *period*, and *communication mode*. The communication mode is used to distinguish between *producers* (i.e., service providers) and *consumers* (i.e., service subscribers). The task must also declare whether the communication is meant to be intra- or inter-ECU.

AP2 Deregistration phase. When leaving AP-LET, a task must specify its *task id*, so that AP-LET can proceed with deregistering it from all the subscribed services (if the task is a subscriber) or removing the provided service itself (if the task is a producer).

AP3 Intra-ECU communications. Given an arbitrary ECU $E_s \in \mathcal{E}$, the intra-ECU communications involving a producer task $\tau_p \in \Gamma_s$ and a consumer $\tau_c \in \Gamma_s$ follow the intra-ECU communication protocol (see Sec. IV-C).

AP4 Inter-ECU communications. Given an arbitrary ECU $E_s \in \mathcal{E}$, the inter-ECU communications involving a producer task $\tau_p \in \Gamma_s$ and a consumer $\tau_c \in \Gamma \setminus \Gamma_s$ follow the inter-ECU communication protocol. The inter-ECU communication protocol can be instantiated by either HP-AP-LET or TM-AP-LET (see Sec. IV-D and Sec. IV-E).

AP5 LET communication invariant. The intra-ECU communication protocol needs to guarantee that an intra-ECU communication involving a job $\tau_{p,g}$ of producer task $\tau_p \in \Gamma_s$ and a job $\tau_{c,a}$ of a consumer task $\tau_c \in \Gamma_s$ for a service $\theta_j \in \Theta$ follows the following invariant. If $\tau_{p,g}$ and $\tau_{c,a}$ are released at times t_o and t_r, respectively, the message sent by $\tau_{p,g}$ is said to be *valid* for $\tau_{c,a}$ only if $t_r \geq t_o + T_p$. The most recent valid one must be considered if there are multiple valid messages at a time.

AP6 SL-LET communication invariant. Both HP-AP-LET or TM-AP-LET ensure that an inter-ECU communication involving a job $\tau_{p,g}$ of producer task $\tau_p \in \Gamma_s$ and a job $\tau_{c,a}$ of a consumer task $\tau_c \in \Gamma_z \neq \Gamma_s$, for a service $\theta_j \in \Theta$, follows the following invariant. If $\tau_{p,g}$ and $\tau_{c,a}$ are released at times t_o and t_r, respectively, the message sent by $\tau_{p,g}$ is said to be *valid* for $\tau_{c,a}$ only if $t_r \geq t_o + T_p + \delta_{s,z} + \omega^j_{s,z}$. The most recent valid one must be considered if there are multiple valid messages at a time.

Note also that rules **AP5** and **AP6** establish two invariants required to comply with LET (intra-ECU) and SL-LET (inter-ECU) for guaranteeing data-flow determinism. **AP5** states that consumers need to read the most recent message sent by a job of the producer at the end of its period. **AP6** extends AP5 to account for the clock synchronization error $\delta_{s,z}$ between two different ECUs and the worst-case transmission time $\omega^j_{s,z}$.

Note that dynamic task handling can lead to a transitory interval of non-deterministic behavior, due to computations required by registration/deregistration operations.

Consider, for example, a linear task chain composed of three tasks $\tau_1 \succ \tau_2 \succ \tau_1$, where $\succ$ states the data dependencies between tasks. The registration operation can extend or register a new service and, hence, fork the existing cause-effect chain to provide multiple different outputs $\tau_1 \succ \tau_2 \succ \tau_3$ and $\tau_1 \succ \tau_2 \succ \tau_4$. After a transitory time the registration operation is performed by the HP-LET tasks, the deterministic behavior is restored in the steady state. It is possible to use the registration operation also to enhance the cause-effect chain by adding a task in the middle of the chain $\tau_1 \succ \tau_2 \succ \tau_3 \succ \tau_4$. However, to not break determinism and the functional behavior of the cause-effect chain, the tasks that must link to the newly registered task must deregister to their actual service and perform a new registration to the new one.

The same concept applies also for the deregistration phase, which can be used to remove an existing service from the non-linear cause-effect chain by deregistering one of the last tasks of the chain $\tau_1 \succ \tau_2 \succ \tau_3$. After the deregistration operation performed by the HP-LET tasks, the deterministic behavior is restored. It is possible to use the registration operation also to reduce the cause-effect chain by removing a task in the middle of the chain. However, to not break determinism and the functional behavior of the cause-effect chain, the task must unlink from the deregistering task and perform a new registration.

C. Intra-ECU communications

This section focuses on the communications occurring within tasks in an arbitrary ECU $E_s \in \mathcal{E}$ (rule **AP3**).

For these communications, following previous work [26], we handle intra-ECU (but inter-core) communications while guaranteeing adherence to the original LET semantics [5] by leveraging tasks running at the maximum priority, which are referred to as HP-LET tasks in the following.

The protocol is defined by the following set of rules:

IE1 For each core $c_{s,k} \in \mathcal{C}_s$, an HP-LET task $\tau^{\mathrm{L}}_{s,k} \in \Gamma_{s,k}$ is introduced and runs at the highest priority to manage intra-ECU communications within $E_s \in \mathcal{E}$.

IE2 When a task joins AP-LET for intra-ECU communications on a core $c_{s,k} \in \mathcal{C}_s$ of ECU $E_s \in \mathcal{E}$, it registers to the HP-LET task $\tau^{\mathrm{L}}_{s,k} \in \Gamma_{s,k}$ of E_s.

IE3 When a task quits AP-LET on a $c_{s,k} \in \mathcal{C}_s$ of ECU $E_s \in \mathcal{E}$, it de-registers to the HP-LET task $\tau^{\mathrm{L}}_{s,k} \in \Gamma_{s,k}$ of E_s.

IE4 For each core $c_{s,k} \in \mathcal{C}_s$, consider the job of a task $\tau_i \in \Gamma_s$ running in $[t_o, t_o + T_i)$. During its execution, the job reads and writes data from/to a local variable, which is accessible by both τ_i and the LET task $\tau^{\mathrm{L}}_{s,k}$ of its core. All the read operations of such a job are performed by $\tau^{\mathrm{L}}_{s,k}$ and are scheduled to occur at the beginning of the interval $[t_o, t_o + T_i)$, i.e., at t_o. All write operations of such a job are performed by $\tau^{\mathrm{L}}_{s,k}$ and are scheduled to occur at the beginning of the interval $[t_o + T_i, t_o + 2 \cdot T_i)$ in which the next job runs, i.e., at $t_o + T_i$.

IE5 For each core $c_{s,k} \in \mathcal{C}_s$, task $\tau^{\mathrm{L}}_{s,k}$ is activated at the release time of each task that needs to produce or consume data for/from tasks in the same ECU, and behave

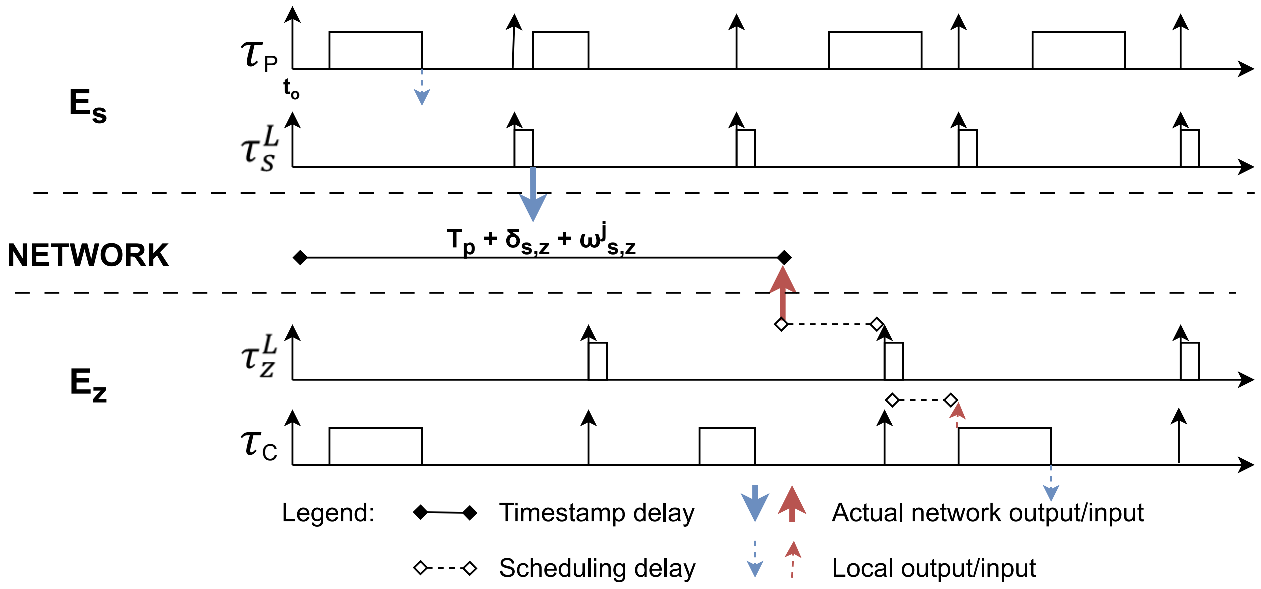

Fig. 2: Data flow between producer τ_p and a consumer τ_c under the HP-AP-LET protocol. Local outputs/inputs are made available to the HP-LET task by producer/consumer tasks at the end of their period. Instead, network outputs and inputs are managed by the HP-LET task at the beginning of activation period of every job of τ_p and τ_s, respectively. All communication operations are performed at pre-determined time instants.

following rules **IE2-IE4**[2].

IE6 Given an arbitrary time t, let $\Gamma^L_s(t)$ be the set of LET tasks (in different cores $c_{s,k} \in \mathcal{C}_s$) that need to perform read and write operations at time t in the ECU $E_s \in \mathcal{E}$ under analysis. Each $\tau^{\mathrm{L}}_{s,k} \in \Gamma^L_s(t)$ starts first executing all the write operations scheduled for time t.

IE7 After the writing phase, all LET tasks $\tau^{\mathrm{L}}_{s,k} \in \Gamma^L_s(t)$, synchronize before starting the reading phase.

IE8 Finally, all LET tasks $\tau^{\mathrm{L}}_{s,k} \in \Gamma^L_s(t)$ perform all the read operations scheduled for time t.

Rules **IE6-IE8** guarantee ensuring the so-called *"first all writes - then all reads"* LET semantics [5, 9] required by scheduled communications in a multi-core platform (rule **IE6** and **IE8**), taking into account that communications occurring in different cores can have different duration (rule **IE7**). Furthermore, the LET communication invariant **AP5** is satisfied by writing (and reading) each message in the next period of the producer (rule **IE4**).

D. The HP-AP-LET Protocol

The first approach to implement SL-LET on AUTOSAR Adaptive is through dedicated tasks running at the highest priority. To integrate SL-LET with the LET protocol discussed in the previous section, it is possible to extend the HP-LET tasks (see Sec. IV-C) to also implement inter-ECU communications. They can hence send/receive messages to/from the network on behalf of producers and consumers executing on the same ECU, similarly to the intra-ECU communication protocol.

The rules of HP-AP-LET are reported next, specializing those of the AP-LET meta-protocol.

HP1 For each ECU $E_s \in \mathcal{E}$, the HP-LET tasks $\tau^{\mathrm{L}}_{s,k}$ of each $c_{s,k} \in \mathcal{C}_s$ also manage inter-ECU communications registered in E_s.

HP2 When a new task joins AP-LET for inter-ECU communications on a core $c_{s,k} \in \mathcal{C}_s$ of ECU $E_s \in \mathcal{E}$, it registers to the HP-LET task $\tau^{\mathrm{L}}_{s,k} \in \Gamma_{s,k}$.

HP3 When a task quits AP-LET on a core $c_{s,k} \in \mathcal{C}_s$ of ECU $E_s \in \mathcal{E}$, it de-registers to the HP-LET task $\tau^{\mathrm{L}}_{s,k} \in \Gamma_{s,k}$.

HP4 On the core $c_{s,k} \in \mathcal{C}_s$ of the producer task, the communication is handled by the HP-LET task $\tau^{\mathrm{L}}_{s,k}$ executing on the same core of the producer. $\tau^{\mathrm{L}}_{s,k}$ is activated at the release time of each job of each registered task to perform communications. $\tau^{\mathrm{L}}_{s,k}$ also updates the timestamp of the message setting it to the current activation time plus synchronization delay plus WCTT, i.e., $t_o + T_p + \delta_{s,z} + \omega^j_{s,z}$, with t_o defined as in rule **AP6**.

HP5 On the core $c_{z,x} \in \mathcal{C}_z$ of the consumer task $\tau_c \in \Gamma_z$, messages are received by the HP-LET task $\tau^{\mathrm{L}}_{z,x}$ executing on the same core of the consumer. $\tau^{\mathrm{L}}_{z,x}$ runs at the release time of each job of each registered task, and it processes the received message for τ_c when its next job $\tau_{c,a}$ is released, say at t_r. $\tau^{\mathrm{L}}_{z,x}$ then delivers the message to the consumer task only if t_r is $\geq$ to the message timestamp, i.e., if it satisfies the invariant **AP6** (see Sec. IV-B).

[2]Please note that this rule does *not* specify a strictly-periodic activation. Indeed, the HP-LET task $\tau^{\mathrm{L}}_{s,k}$ follows the activation pattern Generalized Multiframe Task [27]. This is further discussed in Section V .

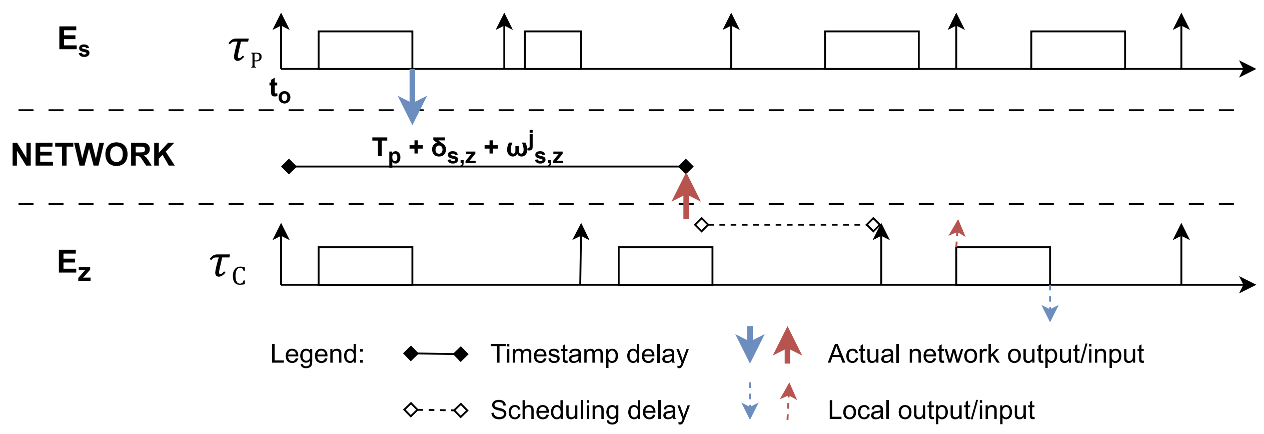

Fig. 3: Data flow between producer τ_p and a consumer τ_c under the TM-AP-LET protocol. As opposed to the HP-AP-LET approach, network outputs and inputs are directly managed by producer and consumer tasks, respectively. Furthermore, all communication operations are scheduler-dependent, while data-flow determinism is ensured by using timestamps.

Example. Figure 2 shows an example for the HP-AP-LET protocol. Consider a producer task $\tau_p \in \Gamma_{s,k}$ and a consumer task $\tau_c \in \Gamma_{z,x}$ executing respectively on the E_s and E_z and communicating using service θ_j. When τ_p finishes its execution, it makes available the output to the HP-LET task $\tau^{\mathrm{L}}_{s,k}$, whose activation is set at the beginning of the next release time of τ_p, when it sends the message from the network. Note that the time τ_p makes the output available may vary for each job. However, rule **HP4** enforces the output operation to occur at the end of the period of the current job of τ_p and sets the timestamp of the message to $t_o + T_p + \delta_{s,z} + \omega^j_{s,z}$ as if it always experiences a fixed network delay. Upon receiving the message on E_z, the HP-LET task $\tau^{\mathrm{L}}_{z,x}$ then forwards it to $\tau_c \in \Gamma_{z,x}$ according to the rule **HP5**, enforcing determinism.

E. The TM-AP-LET Protocol

Differently from the HP-AP-LET, TM-AP-LET does not use high-priority tasks to manage SL-LET communications. Instead, timestamps are used to enforce data-flow determinism at the cost of losing time determinism, but with the advantage of reducing overheads and high-priority interference. Pros and cons of the two approaches are detailed in Sec. V-D. The rules of the TM-AP-LET protocol are listed next, extending those of the meta-protocol.

TM1 When a task joins AP-LET, it registers directly to the AUTOSAR Adaptive services it requires.

TM2 When a task quits AP-LET, it de-registers directly from the AUTOSAR Adaptive services it was using.

TM3 When a job $\tau_{p,g}$ of a task $\tau_p \in \Gamma_s$, released at time t_o, produces a message, it directly performs the output operating by sending it through the network. Before sending the message, the timestamp value of the message is updated with the current activation time plus the synchronization delay plus its WCTT, i.e., setting it to $t_o + T_p + \delta_{s,z} + \omega^j_{s,z}$.

TM4 On the ECU $E_z \in \mathcal{E}$ of the consumer task $\tau_c \in \Gamma_Z$, messages are directly received by the consumer. At the beginning of the execution of each job $\tau_{c,a}$ of τ_c, released at t_r, only the most recent message with timestamp less or equal to t_r is made available to τ_c, i.e., only if it satisfies the invariant **AP6** (see Sec. IV-B). If the invariant is not satisfied, it is checked again in the next job.

The key difference between the two approaches is when the data is logically produced (which in this case means "made available outside the ECU") and thus made available to the consumer: in the HP-AP-LET approach the corresponding time instants are entirely deterministic since the HP-LET tasks run at pre-determined time instants, as reported in Sec. V-A. Differently, under TM-AP-LET, messages are directly sent and received by the producer and consumer tasks, which are subject to scheduling-induced delays as they do not run at the highest priority. Hence, while data-flow determinism is guaranteed by construction by invariant **AP6**, time determinism cannot be guaranteed.

Example. Figure 3 shows an example of the TM-AP-LET protocol in action. Consider a producer task $\tau_p \in \Gamma_{s,k}$ and a consumer task $\tau_c \in \Gamma_{z,x}$ executing on E_s and E_z, respectively, and communicating using service θ_j. When τ_p completes its execution, it sends the output directly to the network. Despite the time needed by τ_p to send the output may vary, Rule **TM3** sets the timestamp of the message to $t_o + T_p + \delta_{s,z} + \omega^j_{s,z}$ as if the output is sent at the end of τ_p's period (accounting for communication delay and synchronization). Upon receiving the message on E_z, τ_c considers the message valid if it fulfills **AP6**. As opposed to the HP-AP-LET protocol, TM-AP-LET guarantees only data-flow determinism, as the time the producer sends the output to the network is not deterministic. However, communication does not involve high-priority tasks.

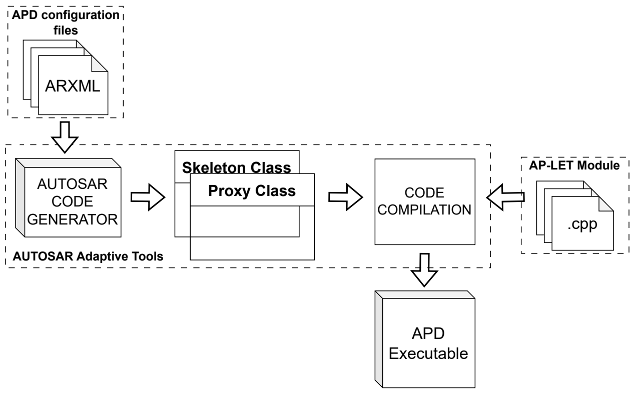

Fig. 4: Workflow to obtain the AP-LET Adaptive Platform executable.

V. Implementation

This section describes the implementation of our solution. The workflow is shown in Fig. 4. The implementation was performed in C++ on the Linux operating system, leveraging the AUTOSAR Adaptive Platform Demonstrator (APD), which is freely available to all members of the AUTOSAR consortium. APD, in turn, was modified to implement the proposed protocols. Following AUTOSAR Adaptive, services are statically declared within ARXML files. Starting from the ARXML, the service-specific routines and data structures needed for AP-LET are generated by modifying the AUTOSAR Adaptive tooling, which generates the service-specific classes. These classes are Proxy and Skeleton, illustrated in Fig. 4, and constitute wrappers of AUTOSAR Adaptive to handle communications. In particular, the Skeleton class implements all the operations that need to be performed on the sender side (e.g., publishing messages and offering services). Similarly, the Proxy class implements all the operations on the receiving side, such as the subscriptions to services and the reception of messages. The generated classes are merged with our custom AP-LET module for Adaptive to generate AP-LET-aware executables for AUTOSAR Adaptive.

To implement the service-oriented architecture, we used the SOME/IP publish/subscribe middleware, which is the default solution in Adaptive. Therefore, inter-ECU communications are performed by wrapping the AUTOSAR Adaptive API to interact with SOME/IP into a unified interface for both intra-ECU and inter-ECU communications (**RQ3**).

The clock between different ECUs on AUTOSAR Adaptive is regulated using the time synchronization functionality provided by the Time Synchronization functional cluster [28].

Next, we describe our protocol-specific implementations.

A. LET and SL-LET with High-Priority Tasks

We present how we managed LET and SL-LET with high-priority tasks. This is common to both the intra-ECU protocol (Sec. IV-C) and the HP-AP-LET protocol for inter-ECU communications (Sec. IV-D).

Data Structures. To manage the communications between producers and consumers by means of high-priority tasks, logical communication channels are created. They are meant to transfer the data needed to manage the services used by producers and consumers. Services are implemented in such a way that both local and remote consumers can subscribe to the same service, i.e., to allow a service to support both intra-ECU and inter-ECU communication to different subscribers. Logical communication channels include a metadata section containing all the relevant information related to tasks registered to the channel and buffers to exchange messages between tasks.

Metadata consists of **(1)** a unique **channel identifier**, **(2)** a **status** flag, which states whether new data is available from the producer, **(3)** a **timestamp**, stating when the consumer is ready to accept new data, **(4)** the **size** of the channel's messages, **(5)** the number of **members** joining the channels. To manage communication between tasks, metadata also contains **(6) a reference to the producer task**, **(7)** the **core** where it is executing, and **(8) consumers' references queue**, to address all the consumers of the channel correctly. Finally, to discriminate between intra- and inter-ECU communication it includes **(9)** a flag indicating the **scope** of the communication is used, along with a **(10) pending queue** used by the HP-LET task to store incoming messages from the network, ordered by timestamp.

The logical communication channel also includes a **private memory buffer**, where the message is stored. It can be accessed only by the HP-LET task managing the communication during the communication phases.

Registering to a service. Since AUTOSAR Adaptive relies on a dynamic POSIX-compliant OS, all the tasks running on each ECU $E_s \in \mathcal{E}$ must register before communicating. This operation is required to allow tasks to join AP-LET at runtime.

On each arbitrary consumer ECU $E_s \in \mathcal{E}$ and core $c_{k,s} \in \mathcal{C}_s$, the HP-LET tasks $\tau^{\mathrm{L}}_{s,k} \in \Gamma_{s,k}$ keeps track of the tasks registered to AP-LET. The registration phase allows producer and consumer tasks to share the information required by HP-LET tasks to handle communication. Every time a task τ_i registers to AP-LET to either publish or subscribe for a topic, a *registration data structure* is created. It is mapped to a memory segment shared between τ_i and the corresponding HP-LET running on its core. For each registering task τ_i, the data structure's metadata contains:

(1) the **task name** is used as a unique identifier, **(2)** the last **activation time**, **(3)** the **period**, **(4)** the **core** where it is executing, **(5) service name** to identify the service to which the task is registering uniquely, **(5)** the **communication mode** for a service (producer or consumer), **(6)** the **shared buffer** used to communicate with the HP-LET task, **(7)** a **communication scope** flag stating whether the service is internal or external the ECU, and **(8)** a **service ID** in the latter case. During the registration phase, any registering task τ_i also creates a shared-memory buffer to host data. The latter is used by both τ_i and the HP-LET tasks to communicate. Finally, τ_i notifies the HP-LET task executing on the same core to complete the registration. The HP-LET task in turn finalizes the registration phase using the information stored in the shared registration data structures.

Algorithm 1 HP-LET activation time setting process

```
1: $Q$ is the sorted queue managed by the HP-LET task
2: $t$ is the current time
3: $k$ is the core index where HP-LET task is executing
4: function SetNextActivationTime($t$)
5:     for each $\tau_i \in \Gamma_L^k(t)$ do
6:         Insert $(t + T_i)$ in $Q$, sorted in ascending order
7:     return first element in $Q$
8: end function
```

Each HP-LET task must then activate each time a task registered to AP-LET is released. This mechanism is implemented using a private ordered queue that keeps track of the next activation time of each task registered to AP-LET on the same core. Algorithm 1 shows how the HP-LET activation time is selected, where $\Gamma_L^k(t)$ denotes the set of tasks registered to AP-LET running on the k-th core that have an activation event at time t. For each task τ, the new activation time is computed by adding the related period T_τ to the current time t (line 6), which is eventually inserted in the ordered queue Q managed by the HP-LET task in ascending order. Finally, the next activation time of the HP-LET task is chosen by taking the first element in Q. The schedulability analysis of the HP-LET task can be performed by leveraging the multi-frame task model, as discussed in [9]. Finally, the deregistration operation is performed at runtime by tasks in order to leave the AP-LET protocol. Once a task is deregistered to AP-LET, the shared buffer used for communication with the HP-LET task is deleted. Then the task is unlinked with all the communication channels it was connected to, and its registration data structure is deleted.

Implementation of Intra-ECU communications. Intra-ECU communication is managed by means of three types of buffers: **(i)** a buffer for the producer, **(ii)** a buffer of the communication channel, and **(iii)** a buffer for each consumer. (i) and (iii) are the shared buffers specified by each task during the registration phase. When an output operation is managed by the HP-LET task related to a certain producer, the HP-LET task copies the data produced by the producer in buffer (i) to buffer (ii). In general, when producer and consumer tasks execute on the same core, it is possible to employ pointer swapping of buffer (i) and buffer (iii), bypassing buffer (ii) and avoiding unnecessary memory copy operations [29]. Similarly, when an input operation is processed, the HP-LET task running in the core of each consumer copies the data from buffer (ii) to buffer (iii). Each consumer's memory buffers are accessed by the *consumers references queue*, presented in Sec. V-A. Similarly, the buffer of the consumer is accessed through the *producer reference*. Synchronization between LET tasks is achieved by using a custom barrier based on POSIX spinlocks. As done in [9, 10, 26], we consider LET intervals that never overlap to avoid inconsistencies.

Implementation of Inter-ECU Communications. In AUTOSAR Adaptive, inter-ECU communications occur through the Proxy and Skeleton classes, which interact with the underlying SOME/IP layer. These classes are automatically generated from the ARXML configuration files, which were extended to support inter-ECU communication. Skeleton contains the API to offer services and send messages. Similarly, Proxy contains the functions to subscribe to services. In our implementation of AP-LET, each core that performs inter-ECU communication interacts with an instance of Skeleton and Proxy. SOME/IP provides two types of communication: *synchronous* and *asynchronous*. The former forces the read of messages to occur upon reception on the destination ECU. Since AP-LET requires messages to be read at predetermined time instants, this work uses asynchronous communication. Under HP-AP-LET, HP-LET tasks manage Skeleton and Proxy on behalf of all producer/consumer tasks, while under TM-AP-LET, Skeleton and Proxy are directly managed by application tasks depending on the used interfaces.

B. HP-AP-LET implementation

Next, we focus on HP-AP-LET, summarizing the operations occurring on the producer and consumer side.

Producer side. Every time an inter-ECU communication message needs to be sent, independently from the service it belongs to, the producer writes the message in the buffer shared with the HP-LET task created during the registration phase. Messages are actually sent over the network by the HP-LET task only at the next release time of the producer task, following rule **HP4**. Before sending the message, however, the message timestamp is updated with the next activation time w.r.t. the job that sent the message, with an additional delay equal to the WCTT plus the synchronization error. It is important to remark that the producer is not aware of the presence of the HP-LET task and can transparently send messages using a unified API (requirement **RQ4**).

Consumer side. Every time a read operation is executed by the HP-LET task, the possible messages to be delivered are considered. These messages are given by those contained in the queue of pending messages of the channel, defined in Sec. V-A, which contains all the messages that have been received in previous executions of the HP-LET task but have not yet been delivered (because **AP6** was not satisfied). Among all these messages, if there is at least one message satisfying **AP6**, the most recent one is delivered to the consumer, and all the others are discarded. Otherwise, the messages contained in the queue of the Proxy class are moved to the queue of pending messages of the channel and no message is delivered to the consumer. Delivered messages are copied to the buffer of the HP-LET task of the consumer's core. Also in this case, the consumer is not aware of the presence of the HP-LET tasks.

C. TM-AP-LET implementation

Registration Phase. In this case, the registration phase is much simpler and only requires registering for the services of interest by using the standard API of AUTOSAR Adaptive.

This simplicity is due to the fact that TM-AP-LET does not involve other tasks in the communication phase. For each task registering as a subscriber for a service, a local message queue is created and used to store received messages.

Producer side. Every time the producer needs to send a message on a service, it must set the message timestamp according to the protocol rules (see Sec. IV-B, **AP6**). This operation is performed by a function introduced into the `ara::com` functional cluster.

Consumer side. At the beginning of the consumer's execution, the possible messages to be used are processed, similarly to Sec. V-B. They are those stored in the local message queue (containing messages that were not delivered in previous jobs because **AP6** was not satisfied) and those received by the network since the last job execution. The latter messages are contained in a queue of Proxy. For each of these messages, the timestamp is evaluated according to **AP6**. If there is at last one valid message, i.e., a message for which the timestamp value precedes the current activation time of the consumer, the most recent valid message is used by the current job of the consumer, and all the others are discarded. Otherwise, no new message is delivered to the current job. These operations were implemented within the `ara::com` functional cluster.

D. *Qualitative comparison of the two protocols*

Both HP-AP-LET and TM-AP-LET have advantages and disadvantages. HP-AP-LET has the advantage of guaranteeing both time and data-flow determinism of communications, which are performed by tasks running at the highest priority and ensures time determinism of communications. Nevertheless, since HP-LET tasks are scheduled with the highest priority of the systems, some overheads are introduced. This means that whenever it is required to perform communication operations, the current task is preempted, and a context switch takes place. Such context switch introduces an overhead in the order of microseconds that, depending on the number of communicating tasks, their periods, and the size of messages, could become non-negligible and affect the overall system performance and the response time of all tasks due to interference. Furthermore, higher-priority functional tasks need to wait for the termination of communications of low-priority tasks, which are performed at the highest priority by the HP-LET task. This leads to an SL-LET-induced priority-inversion phenomenon [30]. Hence, more communication operations may come at the cost of more priority inversion and more overhead introduced.

Conversely, TM-AP-LET does not introduce these drawbacks, as it does not need additional tasks for performing the communication over the network. However, HP-LET tasks are still present for intra-ECU communication. This design choice is made because it allows to perform intra-ECU communications more efficiently, without involving the network stack, while maintaining their design coherent with well-established industrial practices in automotive systems [9, 29]. Furthermore, this guarantees time-deterministic intra-ECU communications. While both our protocols leverage message timestamps, HP-AP-LET can be optimized similarly as suggested by [11] to reduce the space overhead in the message payload by using a sequence number rather than an actual timestamp. This optimization is, instead, not possible for TM-AP-LET. Hence, if this optimization is applied to HP-AP-LET, TM-AP-LET needs more room in the message payload to store the needed timing information: on 64-bit ARM Linux machines, for example, 16 bytes are needed for storing a `timespec` data structure versus one or a few bytes needed to store sequence numbers. With respect to HP-AP-LET, the overhead is therefore introduced in the message size rather than in the CPU execution time. Moreover, TM-AP-LET can only guarantee data-flow determinism and not time determinism since sending and receiving operations can occur at any time during the execution of the tasks. Instead, the data-flow determinism is guaranteed since it only depends on the time when the messages are read. Also, causality is preserved.

As highlighted in [11], the adoption of the LET paradigm for distributed systems comes with different limitations. Although it is possible to embed the communication delay into the logical execution time of the sender task, this technique leads to serious disadvantages. In particular, the logical execution time of the sender task must be sufficiently large to include both the task response time and the communication delay, bounding the communication delay to not exceed the logical execution time of the sender task. This restriction is usually hard to meet. The AP-LET protocol overcomes this limitation by addressing the communication delay to the message validity time, which is represented by its timestamp. However, the time- and data-flow determinism come with the price of an inflated maximum end-to-end latency due to writing operations performed at the end of the period.

E. *Robustness of AP-LET protocol*

Although AP-LET is designed to enable determinism in the AUTOSAR Adaptive system, operations performed at runtime may threaten the system's deterministic behavior. Such operations are represented by dynamic task handling to join and leave the LET paradigm.

To better explain the effect of the dynamic registration and deregistration operations on the determinism of the AP-LET protocol, it is required to discriminate whether the communication is meant to be inter- or intra-ECU.

Intra-ECU. As stated by rules **IE2** and **IE3** in Sec. IV-B, the registration/deregistration of a task to the AP-LET meta-protocol for intra-ecu communication is handled by the high-priority tasks. Hence, during the registration/deregistration phase, the high-priority task creates/destroys the communication channels and shared buffers related to the task, as explained in Sec. V-A, updating the cause-effect chain. Rules **IE4** and **IE5** state that input (read) and output (write) operations are performed by the high-priority tasks at predetermined time instances. Therefore, any changes on the cause-effect chain lead to an initial transitory interval of time of non-deterministic behavior, due to computations required by registration/deregistration operations. After that, the deterministic behavior is ensured by rules **IE4** and **IE5**.

Inter-ECU. For the HP-AP-LET protocol, as stated by rules **HP2** and **HP3**, tasks that want to join/leave the AP-LET protocol must register/deregister to the HP-LET tasks, which then take charge of registering/deregistering to AUTOSAR Adaptive services. Rules **HP4** and **HP5** state that input (receive) and output (send) operations are performed by the high-priority tasks at predetermined time instances. Therefore, any changes on the cause-effect chain lead to an initial period of non-deterministic behavior, due to computations required by registration/deregistration operations. After that, the deterministic behavior is ensured by rules **HP4** and **HP5**. For the TM-AP-LET protocol, rules **TM1** and **TM2** state that tasks register and deregister directly to the AUTOSAR Adaptive services, while rule **TM4** ensures that data-flow determinism is guaranteed even in the presence of changes in the cause-effect chain.

From this analysis, it is possible to infer that dynamic registration and deregistration operations do not harm the deterministic behavior in the steady state of the system and the possibility of bounding the end-to-end latency.

VI. Evaluation

The evaluation of this paper consists of two different experimental campaigns based on two different realistic automotive applications. The first one is based on the WATERS 2017 Challenge by Bosch and evaluates the overhead injected by the HP-LET tasks and how it reflects on response times. The second one is based on the Brake Assistant Demonstrator of Adaptive and shows the benefit of deterministic communication in a distributed environment.

The purpose of the evaluation is the following: **(i)** to show that the overhead cost introduced by implementing HP-AP-LET and TM-AP-LET is limited, and it is a price worth paying for achieving determinism in AUTOSAR Adaptive; **(ii)** comparing, still in terms of overhead, the difference between HP-AP-LET and TM-AP-LET, highlighting what is the extra cost needed by HP-AP-LET to achieve time-determinism (in addition to the data-flow determinism already provided by TM-AP-LET); **(iii)** reporting on the effects on the jitter for both HP-AP-LET and TM-AP-LET to validate their properties empirically; **(iv)** reporting on practical advantages in terms of percentage of dropped frames and inputs mismatches of an automotive communication pipeline.

A. Evaluation based on the WATERS 2017

The evaluation test-bed consists of an automotive application (namely, the one presented in the WATERS Challenge 2017 [13]). This application has been modified to be executed on the AUTOSAR Platform Demonstrator (APD), which is freely available to all members of the AUTOSAR consortium. APD, in turn, has been modified to implement the AP-LET protocols, as discussed in Sec. V. To achieve a meaningful distributed setup required to evaluate AP-LET, the WATERS 2017 application has been split among two Raspberry Pi4 boards, powered by a quad-core 64-bit ARM SoC and running the Linux operating system (kernel version 5.4). The communication leverages the SOME/IP protocol over Ethernet.

Test-bed Application. We tested a large number of different configurations of task-to-ECU and task-to-core allocation and decided to report the two most significant ones. Next, we report the experimental results of both the allocations shown in the Fig. 5. Configuration A) has been designed to stress the system by maximizing the number of messages exchanged over the network to stress the AP-LET protocol variants, thus providing the most interesting results. Indeed, *Task_10ms*, allocated to `ECU2`, communicates with all the other tasks allocated to `ECU1`, thus maximizing the number of inter-ECU communications. Configuration B) balances both computation load and network traffic among the three cores of the two ECUs. One core on each ECU is left unused by the WATERS challenge and is dedicated to all the basic applications and services that AUTOSAR AP needs to run. Since all tasks have a different period, following the naming convention used in the WATERS 2017 Challenge, the name of each task is made by the composition of the word *"Task_"* and its period (reported in Fig. 5). Other tasks are named *"ISR_"* (i.e., interrupt service routines) followed by a unique identifier (again, following the naming convention of WATERS 2017). Tasks priorities have been assigned according to the rate-monotonic algorithm.

WCTT estimation. An empirical analysis of the network latency has been carried out to obtain an empirical bound on the worst-case transmission time to be used during the protocols. The analysis has been performed considering all the sizes of messages exchanged between ECUs. According to our measurements, the largest delay experienced among all the payloads is 6ms. The WCTT has been then set to 9ms, inflating the largest experienced delay with a 50% safety margin. The synchronization delay has been deemed negligible because our implementation leverages the Time Synchronization functional cluster provided by AUTOSAR Adaptive [28].

WATERS 2017 on AUTOSAR Adaptive. To implement the WATERS 2017 Challenge on AUTOSAR Adaptive, several configuration ARXML of AUTOSAR Adaptive files have been customized to configure the system. ARXML files were defined to describe the services (i.e., message exchanges used in the challenge) used by tasks of the WATERS Challenge. These files include all the information related to a specific service, such as the message type (e.g., string, or integer), the message size, and the service name. Also, a manifest file has been defined for the WATERS application. This file describes the main Linux process that creates all the tasks of the WATERS challenge and specifies the scheduler used for executing it (`SCHED_FIFO`, in this case) and its priority. It also specifies which services are used by the application.

Comparison of two approaches. Fig. 6 shows a comparison between HP-AP-LET and TM-AP-LET implementations in terms of the average and maximum execution time of the LET high-priority tasks. In particular, the average execution time is highlighted by the colored bars, while the spikes represent the

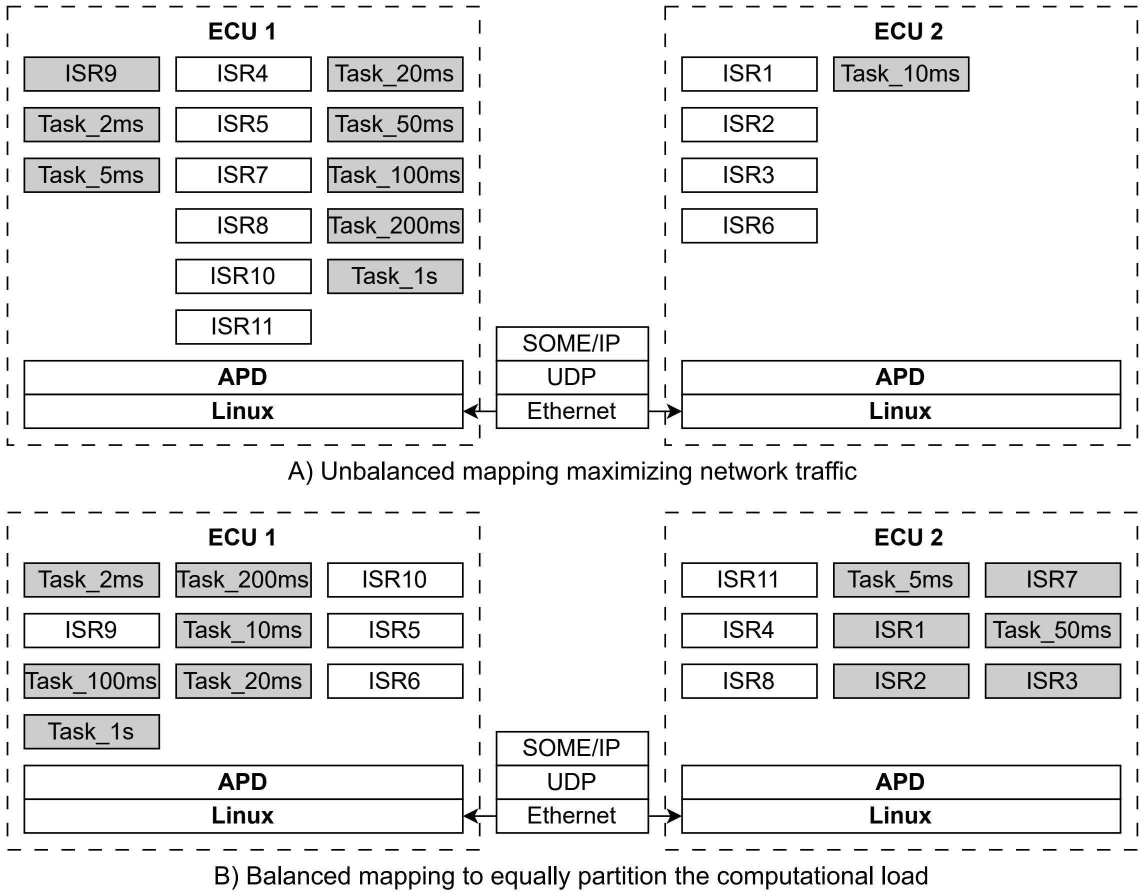

Fig. 5: Task-to-ECU and Task-to-core mapping for the WATERS Challenge 2017. Tasks listed in the same column are allocated to the same core. Tasks in gray color perform inter-ECU communication.

maximum execution time reached during the whole execution of the experiment. Cores 2, 3, and 4 of each ECU contain the tasks shown in the first, second, and third columns in Fig. 5, while core 1 is left unused on both ECUs.

As expected, in both configurations, the overhead introduced by the HP-AP-LET is higher than the TM-AP-LET, in which the overhead of the HP-LET tasks can be considered negligible, as they only manage intra-ECU communications. As shown in Fig. 6, the most significant overhead is experienced in core 3 of ECU 2, which executes the tasks with the higher network traffic load on the whole WATERS challenge obtained with the chosen tasks' mapping. In the evaluation of Configuration B), the overhead injected is present more uniformly among the ECUs, except for core 3 of ECU 1, which experiences the higher overhead. Also, in this case, the higher network traffic is located in the core with the highest overhead, suggesting a strong proportional relationship between these two factors. To better understand the behavior of the HP-AP-LET implementation, further analysis has been carried out considering the most important phases of the HP-LET tasks. These phases are the reads and writes from/to memory buffers and the send operation to the network. The receive operation from the network is not considered because it is performed by the operating system, which copies the received data into the Proxy queue. For this analysis, only the unbalanced mapping configuration has been considered, as it represents the most interesting case between the two configurations, with the higher spikes on maximum overhead.

E	C	Avg Execution Times [μs]			Max Execution Times [μs]		
		W	R	S	W	R	S
1	1	0	0	0	0	0	0
	2	4	63	128	24	275	481
	3	1	0.5	0.5	25	37	25
	4	2	9	72	28	133	346
2	1	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	1	0.4	0.4	28	32	25
	4	3	83	452	28	594	594

TABLE II: Avg. and max. execution times of the different phases of LET communication with HP-AP-LET. Legend: C = Core Number, E = ECU number, W = Writes, R = Reads, S = Sends.

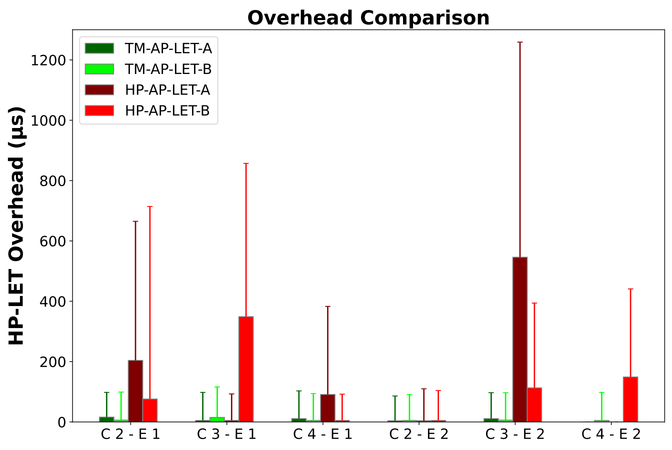

Fig. 6: Overhead Comparison between HP-AP-LET and TM-AP-LET with both unbalanced (-A) and balanced (-B) configurations. Legend: C# = id number of core, E# = id number of ecu.

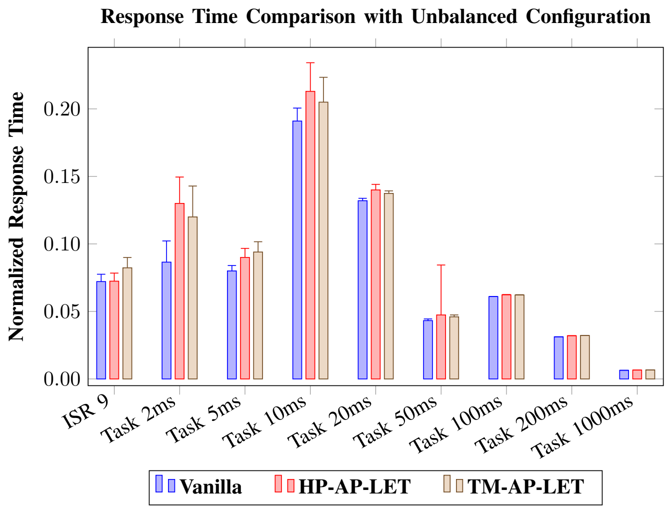

Fig. 7: Average normalized response time with unbalanced configuration under waters vanilla communication compared with HP-AP-LET and TM-AP-LET implementations. Markers highlight the maximum normalized response time observed.

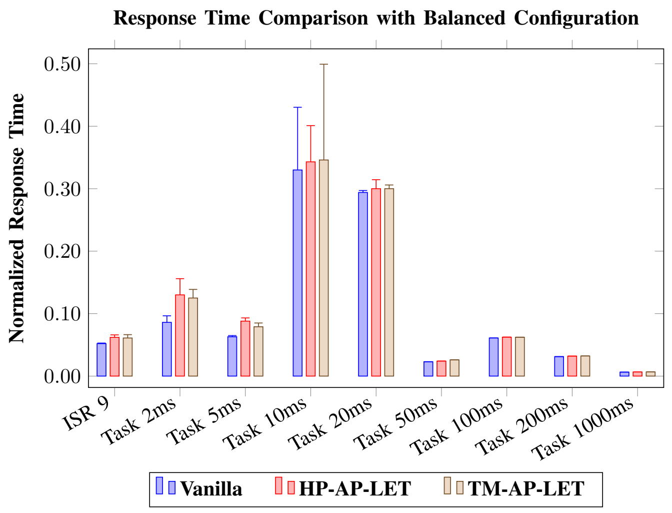

Fig. 8: Average normalized response time with balanced configuration under waters vanilla communication compared with HP-AP-LET and TM-AP-LET implementations. Markers highlight the maximum normalized response time observed.

Table II highlights that most of the overhead is due to the send operations over the network. In the experiment, we observed up to 452 and 594 microseconds for average and maximum registered values, respectively. Since high-priority tasks perform these operations, these overheads can potentially cause a priority-inversion delay since messages of low-priority tasks are managed at the maximum priority, potentially leading to deadline misses. Hence, the TM-AP-LET configuration can be preferred in cases where the network traffic is particularly intense, and the timing constraints of tasks are tight.

Empirical Response Times. Fig. 7 shows the average and maximum response times of nine representative tasks of the WATERS challenge to further compare the proposed implementations with the *vanilla* system and evaluate the impact of AP-LET. The considered vanilla system consists of the standard AUTOSAR Adaptive communications without AP-LET, with communications occurring at any time. The considered tasks are those performing inter-ECU communications. The chart reports the response times normalized by dividing by the corresponding periods. The evaluation considers the unbalanced mapping configuration. Fig. 7 shows that the effect of the priority inversion problem can be noticed. In particular, with the HP-AP-LET implementation, tasks with the highest priority within their core, such as *Task 2ms* and *Task 20ms*, experience a longer response time w.r.t. the TM-AP-LET. This phenomenon occurs since, in the HP-AP-LET implementation, HP-LET tasks can preempt high-priority tasks to perform external communication on behalf of low-priority tasks. In the TM-AP-LET implementation, priority inversion is instead limited to intra-ECU communications, causing a smaller delay. Nevertheless, the phenomena shown in Fig. 7 are expected because the figure targets an unbalanced configuration, which is studied to stress inter-ECU communications. However, with the balanced configuration, priority inversion is drastically reduced, as shown in Fig. 8.

Measured Jitter. Finally, the HP-AP-LET and TM-AP-LET implementations were analyzed in terms of average input and output jitter, considering the same tasks as in Fig. 7. In particular, the output jitter of the k-th job of task τ_i was computed as $J^o_{k,i} = |t_{(k+1)} - t_{s,k}|$, where $t_{(k+1)}$ is the activation time of the (k+1)-th job of τ_i and $t_{s,k}$ is the time in which the actual send operation is performed by the k-th job. The input jitter, instead, was computed as $J^i_{k,i} = |t_r - t_a|$ where t_r is defined in the invariant **AP6** discussed in Sec IV-B, and t_a is the time the message is available to the consumer. The output jitter was normalized to the task's period, while the input jitter was normalized to t_r.

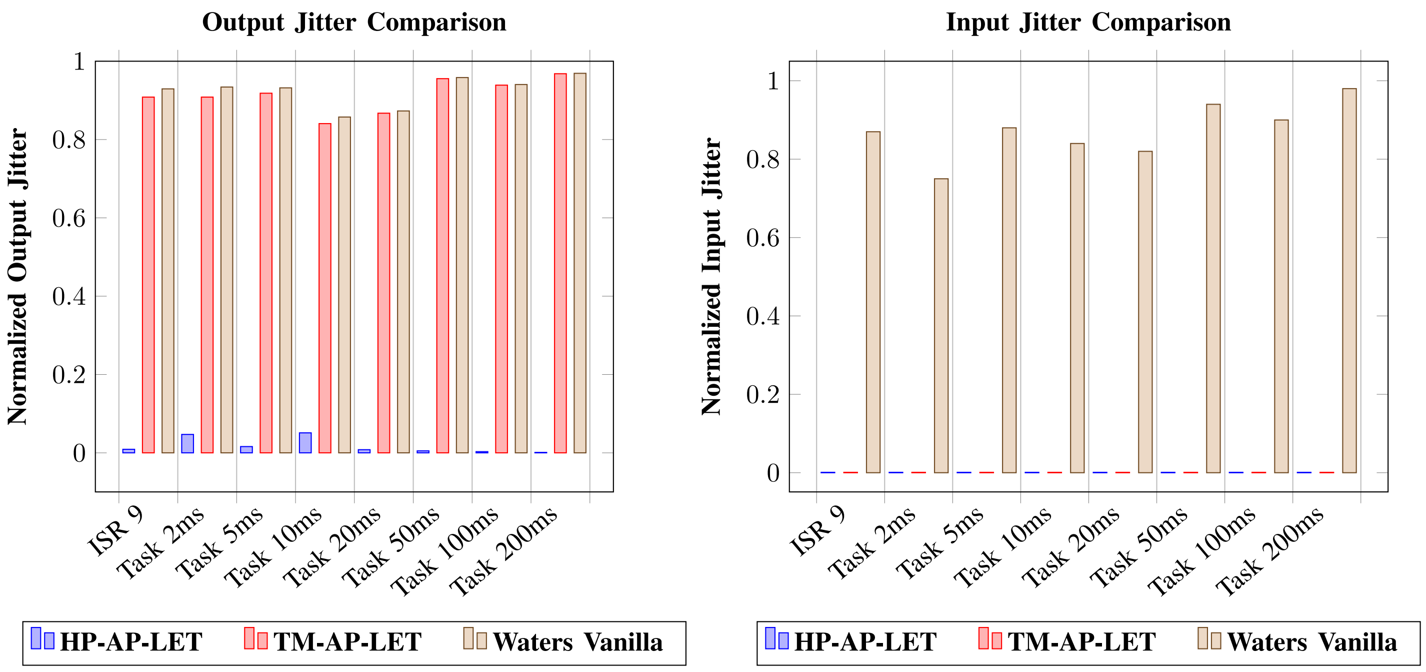

Fig. 9: Average observed normalized output jitter under vanilla communication (no AP-LET) compared with HP-AP-LET and TM-AP-LET.

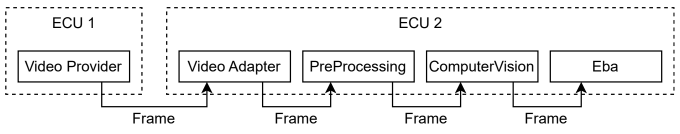

Fig. 10: Brake Assistant application in APD.

S	Frame Error	Percentage on total frames processed (%)			
		VideoAd	PreProc	ComputerVision	EBA
ND	Dropped	0.4	19.5	28.3	19.9
	Mismatch	0	0	18.4	0
D	Dropped	0	0	0	0
	Mismatch	0	0	0	0

TABLE III: Evaluation of non-deterministic execution (vanilla) measuring dropped frames and input mismatch (in percentage) of brake assistant communication pipeline executed in ECU2. Legend: S = Scenario, ND = Non Deterministic, D = Deterministic.

Figure 9 show that the HP-AP-LET implementation removes both input and output jitter by design. Differently, the TM-AP-LET implementation still ensures that invariant **AP6** is satisfied, guaranteeing data-flow determinism and removing the input jitter, while it does not guarantee time-determinism as it is affected by output jitter. The vanilla system without AP-LET shows both input and output jitter.

B. Evaluation based on the Brake Assistant APD

The evaluation test-bed consists of the Brake Assistant application demostrator of APD. As highlighted in [31], it exhibits how non-determinism can potentially have consequences when executing safety-critical applications in AP. The hardware platform is the same as in the previous evaluation. **Test-bed Application.** As shown in Fig. 10, the Brake Assistant application includes five tasks distributed on two ECUs. VideoProvider captures a video frame every 50ms and sends it to VideoAdapter, which has 25ms period. Preprocessing recognizes the lane and sends its bounding box, along with the frame, to ComputerVision, which detects vehicles in the lane and eventually sends the list of detected vehicles to Emergency Brake Assist (EBA). Periods of Preprocessing, ComputerVision, and EBA are 50ms, 50ms, and 25ms, respectively.

Determinism vs Non-Determinism. Two metrics are considered: **(1) Dropped Frames**: Since each producer-consumer pair is based on a single buffer, frames overwritten by the producer before the consumer can read them are considered dropped. **(2) Input Mismatch:** Original frame and lane bounding boxes received by ComputerVision that are not related are considered an input mismatch. Each experiment considered 100k frames. Table III shows both dropped frame and input mismatch in percentage w.r.t. the total amount of frames processed and non-deterministic (Vanilla) scenarios. The communication pipeline under non-deterministic execution suffers from a high rate of dropped frames between the four tasks executed in the second ECU, as well as the input mismatch between PreProcessing and ComputerVision. VideoProvider is not reported in the results, as its purpose is to provide video frames to the communication pipeline. Due to the oversampling, the dropped frames on VideoAdapter are quite low compared to the other tasks in the same ECU. The same behavior is present in the communication between

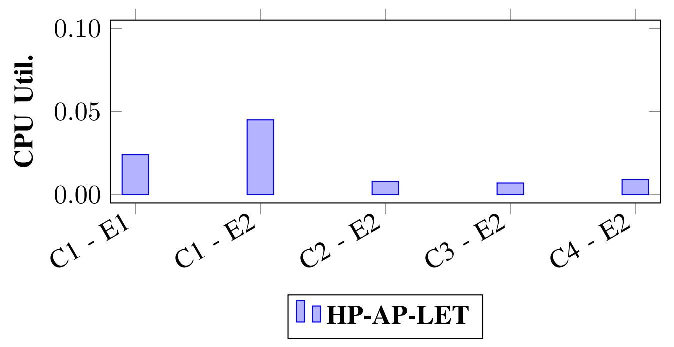

Fig. 11: Overall CPU utilization for each HP-LET task under the HP-AP-LET scenario.

ComputerVision and EBA. However, dropped frames error is propagated through the entire pipeline, causing a higher dropped frame rate in the last task even in the presence of oversampling. The results may vary between executions, as input and output operations occur at scheduling-dependent times. Under the deterministic scenario (HP-AP-LET and TM-AP-LET), the communication pipelines do not suffer from packet loss or input mismatch since data-determinism is preserved in both HP-AP-LET and TM-AP-LET implementations.

CPUs usage. A further evaluation has been performed to measure the CPU utilization overhead under the HP-AP-LET implementation for the Brake Assistant application.

Figure 11 shows the CPU utilization of each HP-LET task computed considering each HP-LET task's overall execution time w.r.t. the whole duration of the experiment. Note that the higher CPU usage is reached in core 1 of both ECUs where HP-LET tasks perform more inter-ECU input/output operations. Nevertheless, the overall overhead, in terms of CPU utilization, is never more than 0.05.

VII. Related Work

Communication mechanisms and middleware for distributed embedded real-time systems received considerable attention in recent years, with works focusing on AUTOSAR Adaptive [31, 32], ROS 2 [12, 33], the DDS [34], and MQTT [35]. Service-oriented communication mechanisms such as those provided by AUTOSAR Adaptive were also studied at the OS level: for example, the message passing mechanism of QNX provides a similar client-server communication infrastructure [36].

Gemlau et al. [37, 38] addressed the challenge of adopting the System-Level LET on high-performance architectures with high data rates, providing a proof-of-concept to optimize the efficiency of the communication stack of the RTE.

The lack of deterministic execution on Adaptive was first highlighted by Menard et al. [31], which also reports a series of inconsistencies occurring in the brake-assistant use-case shipped with the official AUTOSAR Adaptive Demonstrator (APD). Similarly, Köhler et al. [39] proposed the usage of the SL-LET model in Adaptive for determinism. However, it does not discuss AUTOSAR-Adaptive specific protocols for SL-LET and the implementation details. This paper, instead, provides and elaborates design decisions, provides specialized protocols for AUTOSAR Adaptive, and discusses the technical details of the implementation. No other work in the literature proposed protocols for adopting System-Level LET in AUTOSAR Adaptive. Another branch of related papers focuses on the LET paradigm. Many works have been published on this topic: in the following, only those mostly related to this paper are summarized. Partial support for LET was implemented in the rclc Executor from the Micro-ROS project [40]. ROS 2 is a framework being used by about 80% of the automotive OEMs and Tier-1s developing autonomous vehicles [15]. However, its design is too ROS-specific to be applied to the AUTOSAR Adaptive framework, and does not consider SL-LET.

The design of LET on a general-purpose operating system has been investigated in [26]. However, it does not consider SL-LET and AUTOSAR Adaptive, but only POSIX systems.

VIII. Conclusion

This paper presented the design and implementation of the AP-LET meta-protocol, the first protocol tailored to AUTOSAR Adaptive to allow SL-LET communications between tasks in distributed ECUs, providing two different instantiations with different features. An extensive evaluation compared the protocols showing that none of the two dominates the other.

Future work will target the derivation of new protocols for LET-aware message-driven task chains [41] and the development of design-time analysis techniques to estimate the minimal buffer sizes to guarantee no message loss under SL-LET communications.

References

[1] OSEK, "OSEK/VDX Operating System Specification 2.2.3," https://www.osek-vdx.org/index_htm.html, Standard, Feb. 2005.

[2] AUTOSAR, "The AUTOSAR Classic Platform," https://www.autosar.org/standards/classic-platform.

[3] ——, "Explanation of Adaptive Platform Design," https://www.autosar.org/fileadmin/standards/R22-11/AP/AUTOSAR_EXP_PlatformDesign.pdf.

[4] I. S. . TM-2003, "IEEE Standard for Information Technology—Standardized Application Environment Profile (AEP)—POSIX® Realtime and Embedded Application Support," 2003.

[5] T. Henzinger, B. Horowitz, and C. Kirsch, "Giotto: a time-triggered language for embedded programming," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 84–99, 2003.

[6] J. Martinez, I. Sañudo, and M. Bertogna, "Analytical characterization of end-to-end communication delays with logical execution time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2244–2254, 2018.

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

[8] P. Pazzaglia, A. Biondi, and M. D. Natale, "Optimizing the functional deployment on multicore platforms with logical execution time," in *40th IEEE Real-Time Systems Symposium (RTSS 2019)*, 2019.

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

[10] P. Pazzaglia, D. Casini, A. Biondi, and M. D. Natale, "Optimizing inter-core communications under the let paradigm using dma engines," *IEEE Transactions on Computers*, vol. 72, no. 1, pp. 127–139, 2023.

[11] K.-B. Gemlau, L. Köhler, R. Ernst, and S. Quinton, "System-level logical execution time: Augmenting the logical execution time paradigm for distributed real-time automotive software," *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 2, pp. 1–27, 2021.

[12] D. Casini, T. Blaß, I. Lütkebohle, and B. B. Brandenburg, "Response-time analysis of ROS 2 processing chains under reservation-based scheduling," in *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[13] A. Hamann, D. Dasari, S. Kramer, M. Pressler, F. Wurst, D. Ziegenbein., "WATERS industrial challenge 2017." https://waters2017.inria.fr/challenge/#Challenge17.

[14] AUTOSAR, "SOME/IP Protocol Specification," https://www.autosar.org/fileadmin/user_upload/standards/foundation/21-11/AUTOSAR_PRS_SOMEIPProtocol.pdf.

[15] C. Scordino, A. G. Mariño, and F. Fons, "Hardware Acceleration of Data Distribution Service (DDS) for Automotive Communication and Computing," *IEEE Access*, vol. 10, pp. 109 626–109 651, 2022.

[16] S. Kramer, D. Ziegenbein, and A. Hamann, "Real world automotive benchmarks for free," in *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, vol. 130, 2015.

[17] R. Ernst, L. Ahrendts, K.-B. Gemlau, S. Quinton, H. von Hasseln, and J. Hennig, "System level let with application to automotive design," Ph.D. dissertation, TU Braunschweig, 2018.

[18] R. Ernst, L. Ahrendts, and K.-B. Gemlau, "System Level LET: Mastering Cause-Effect Chains in Distributed Systems," in *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, 2018, pp. 4084–4089.

[19] D. de Oliveira, D. Casini, and T. Cucinotta, "Operating System Noise in the Linux Kernel," *IEEE Transactions on Computers*, vol. 72, no. 01, pp. 196–207, jan 2023.

[20] B. B. Brandenburg and M. Gül, "Global scheduling not required: Simple, near-optimal multiprocessor real-time scheduling with semi-partitioned reservations," in *2016 IEEE Real-Time Systems Symposium (RTSS)*, 2016, pp. 99–110.

[21] A. Biondi and Y. Sun, "On the ineffectiveness of 1/m-based interference bounds in the analysis of global EDF and FIFO scheduling," *Real-Time Systems*, vol. 54, no. 3, pp. 515 – 536, Mar 2018.

[22] OMG, "Data Distribution Service (DDS) version 1.4," https://www.omg.org/spec/DDS/.

[23] G. Sciangula, D. Casini, A. Biondi, C. Scordino, and M. Di Natale, "Bounding the Data-Delivery Latency of DDS Messages in Real-Time Applications," in *35th ECRTS 2023*, ser. (LIPIcs), 2023.

[24] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the SymTA/S approach," *IEEE Proceedings - Computers and Digital Techniques*, March 2005.

[25] J. Palencia and M. Gonzalez Harbour, "Schedulability analysis for tasks with static and dynamic offsets," in *Proceedings 19th IEEE Real-Time Systems Symposium*, 1998, pp. 26–37.

[26] D. Bellassai, A. Biondi, A. Biasci, and B. Morelli, "Supporting logical execution time in multi-core POSIX systems," *Journal of Systems Architecture*, vol. 144, p. 102987, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762123001662

[27] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," *Real-Time Systems*, vol. 17, pp. 5–22, 1999.

[28] AUTOSAR, "Specification of Time Synchronization," https://www.autosar.org/fileadmin/standards/R23-11/AP/AUTOSAR_AP_SWS_TimeSynchronization.pdf.

[29] A. Hamann, D. Dasari, S. Kramer, M. Pressler, and F. Wurst, "Communication centric design in complex automotive embedded systems," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[30] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Third Edition*. Springer, 2011.

[31] C. Menard, A. Goens, M. Lohstroh, and J. Castrillon, "Achieving determinism in adaptive AUTOSAR," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020, pp. 822–827.

[32] S. Fürst and M. Bechter, "AUTOSAR for connected and autonomous vehicles: The AUTOSAR adaptive platform," in *2016 46th annual IEEE/IFIP international conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, 2016, pp. 215–217.

[33] H. Teper, M. Günzel, N. Ueter, G. von der Brüggen, and J. Chen, "End-to-end timing analysis in ROS2," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022.

[34] G. Sciangula, D. Casini, A. Biondi, and C. Scordino, "End-to-end latency optimization of thread chains under the dds publish/subscribe middleware," in *2024 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2024, pp. 1–6.

[35] E. Shahri, P. Pedreiras, and L. Almeida, "End-to-end response time analysis for rt-mqtt: Trajectory approach versus holistic approach," in *2023 IEEE 19th International Conference on Factory Communication Systems (WFCS)*, 2023, pp. 1–8.

[36] M. Becker, D. Dasari, and D. Casini, "On the qnx ipc: Assessing predictability for local and distributed real-time systems," in *2023 IEEE 29th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2023, pp. 289–302.

[37] K.-B. Gemlau, J. Peeck, N. Sperling, P. Hertha, and R. Ernst, "A new design for data-centric ethernet communication with tight synchronization requirements for automated vehicles," in *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1. IEEE, 2019, pp. 4489–4494.

[38] K.-B. Gemlau, L. Köhler, and R. Ernst, "Efficient run-time environments for system-level let programming," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 749–754.

[39] L. Köhler, P. Hertha, M. Beckert, A. Bendrick, and R. Ernst, "Robust cause-effect chains with bounded execution time and system-level logical execution time," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 3, pp. 1–28, 2023.

[40] Micro-ROS, "Execution Management," https://micro.ros.org/docs/concepts/client_library/execution_management/.

[41] F. Paladino, A. Biondi, E. Bini, and P. Pazzaglia, "Optimizing Per-Core Priorities to Minimize End-To-End Latencies," in *36th Euromicro Conference on Real-Time Systems (ECRTS 2024)*, vol. 298, 2024, pp. 6:1–6:25.