



HAL
open science

Experimental Validation of User Experience-focused Dynamic Onboard Service Orchestration for Software Defined Vehicles

Pierre Laclau, Stéphane Bonnet, Bertrand Ducourthial, Trista Lin, Xiaoting Li

► To cite this version:

Pierre Laclau, Stéphane Bonnet, Bertrand Ducourthial, Trista Lin, Xiaoting Li. Experimental Validation of User Experience-focused Dynamic Onboard Service Orchestration for Software Defined Vehicles. IEEE ITSC 2024, IEEE ITSS, Sep 2024, Edmonton, Canada. hal-04711357

HAL Id: hal-04711357

<https://hal.science/hal-04711357v1>

Submitted on 26 Sep 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution - NonCommercial - NoDerivatives 4.0 International License

Experimental Validation of User Experience-focused Dynamic Onboard Service Orchestration for Software Defined Vehicles

Pierre Laclau^{1,2}, Stéphane Bonnet¹, Bertrand Ducourthial¹, Trista Lin² and Xiaoting Li²

¹Heudiasyc, CNRS, Université de Technologie de Compiègne, France {firstname.lastname@utc.fr}

²Stellantis, Poissy, France {firstname.lastname@stellantis.com}

Abstract—In response to the growing need for dynamic software features in automobiles, Software Defined Vehicles (SDVs) have emerged as a promising solution. They integrate dynamic onboard service management to handle the large variety of user-requested services during vehicle operation. Allocating onboard resources efficiently in this setting is a challenging task, as it requires a balance between maximizing user experience and guaranteeing mixed-criticality Quality-of-Service (QoS) network requirements. Our previous research introduced a dynamic resource-based onboard service orchestration algorithm. This algorithm considers real-time in-vehicle and V2X network health, along with onboard resource constraints, to globally select degraded modes for onboard applications. It maximizes the overall user experience at all times while being embeddable onboard for on-the-fly decision-making. A key enabler of this approach is the introduction of the Automotive eXperience Integrity Level (AXIL), a metric expressing runtime priority for non-safety-critical applications. While initial simulation results demonstrated the algorithm’s effectiveness, a comprehensive performance assessment would greatly contribute in validating its industrial feasibility. In this current work, we present experimental results obtained from a dedicated test bench. These results illustrate, validate, and assess the practicality of our proposed solution, providing a solid foundation for the continued advancement of dynamic onboard service orchestration in SDVs.

I. INTRODUCTION

As the automotive landscape evolves, the demand for increasingly complex and connected features such as automated driving, advanced infotainment, and Vehicle-to-Everything (V2X) cooperative services is on the rise [1]. Currently, each of these features is implemented as static, monolithic systems that are difficult to update and maintain [2]. Onboard resources are statically allocated before manufacturing and occasionally tweaked through Over-the-Air (OTA) updates, which are limited in scope and frequency [3], [4]. With app stores and feature sets growing, current vehicles are unable to keep pace with user expectations [5].

Traditionally, vehicle differentiation was primarily based on static hardware variants, involving the (de)activation of features in pre-provisioned software components and network allocations [6]. However, the introduction of thousands of optional applications in upcoming app stores requires the implementation of a dynamic mechanism capable of adjusting network and computing allocations in response to the varying preferences unique to each user and resources.

In the future, dynamic service management features are required to extend vehicle capabilities by allocating resources and executing services depending on the current context.

Software Defined Vehicles (SDVs) [7] are emerging as a promising solution as they have the potential to offer dynamic orchestration. They aim to (1) concentrate the execution of features in fewer more powerful Electronic Control Units (ECUs), (2) implement virtualization techniques to enable dynamic resource allocation [8], and (3) set Ethernet as the main network backbone offering high flexibility and reconfigurability with service-oriented communications [9].

However, the dynamic nature of resources poses several challenges, including changes in reserved safety-critical resources based on orchestration, fluctuating network resources in V2X and cloud connectivity, as well as changing user requests that affect resource allocations. Hence, the vehicle must adapt to these changes while respecting the Service Level Agreements (SLAs), which are Quality of Experience (QoE) constraints for users defined by the OEMs, and the user preferences [10]. In this context, the vehicle must dynamically orchestrate services while respecting the SLAs.

In response to these challenges, our previous work [11] proposed to define a user experience (UX) focused runtime priority for non-safety-critical services. We introduced the concept of Automotive eXperience Integrity Level (AXIL), illustrated in Table I, to express user preferences and establish connections with SLAs. This metric allowed us to define a heuristic algorithm running onboard to orchestrate applications while optimizing the *UX-to-resource* ratio of user-focused features. AXIL is similar to the already well-established ASIL levels [12], which help engineers optimize the *safety-to-cost* ratio of safety-critical hardware and software components. In our approach, applications declare runtime modes with varying feature sets, static resource requirements (e.g. CPU, memory, network bandwidth, energy consumption), and associated AXIL scores. When the requested features by the user exceed the available resources, the algorithm selects a mode for each application to maximize the overall UX. This approach allows the vehicle to adapt its current state to the current context (i.e. resources and user requests) while respecting the vehicle SLAs.

Our previous work [11] demonstrated the effectiveness of this approach through simulation results. However, a comprehensive performance assessment would greatly contribute to validating its industrial feasibility. In this current work, we aim to contribute to the field by presenting (1) a dedicated test bench mimicking a typical SDV architecture, (2) realistic test scenarios and validation criteria such as resource usage and network health metrics, and (3) performance results.

The remainder of this paper is organized as follows. Section II presents the related work. Section III summarizes the problem statement and solution presented in [11]. Section IV presents the experimental setup and Section V discusses the experimental results. Section VI concludes the paper.

II. RELATED WORK

The traditional static allocation of resources for onboard applications in vehicles, accompanied by pre-defined rule-based local state management strategies, has long been the norm to ensure a stable and safety-certified software stack [13]. However, the advent of dynamic use cases and updates challenges the feasibility of maintaining control over the combinations of active applications requested by users, vehicles, or road contexts [6]. The conventional rule-based approach falls short in addressing the evolving network traffic requirements to apply context-aware degraded states.

Recognizing this limitation, the automotive industry is transitioning towards dynamic onboard service orchestration leveraging Service Oriented Architectures (SOA) capabilities. This paradigm shift aims to optimize resource usage and reduce engineering efforts for state management [14]. The incorporation of SOA facilitates vehicle-wide monitoring for centralized and context-aware global orchestration [15].

Key to this evolution is the utilization of reconfigurable technologies such as Time-Sensitive Networking (TSN) and Software-Defined Networking (SDN) within the in-vehicle network. These technologies enable dynamic changes in the behavior and resource allocations of onboard software components through reconfigurations [16-18]. Research initiatives propose atomic vehicle-wide reconfiguration strategies employing TSN and SDN to ensure safe transitions, even in dynamic driving scenarios [7], [19], [20]. Examples of global mechanisms enabled by these strategies include improved network intrusion detection [16]. As a result, the focus shifts from manually generating local state management rules to designing a centralized autonomous state management stack.

Additionally, recent research considers bringing containerization to automotive. Containers package applications and their dependencies into an OS-level isolated namespace to achieve hardware-software decoupling and maximum portability. Cloud platforms then distribute containers into clusters of servers according to workload specifications. Dynamic orchestration mechanisms then continuously reconfigure allocations to accommodate for varying workload demand [21].

While current commercial vehicles already implement static containers, applying this additional dynamic layer could bring enhanced use cases, e.g. switching between "autonomous" and "parked" allocations to maximize hardware utilization at all times. However, safety and embedded constraints must be addressed before industrial implementation.

Vehicles host two categories of applications: safety-critical (SC) services and best-effort (BE) applications. SC services require QoS guarantees, including real-time computing and time-sensitive network flows with predefined latency, jitter, and isolation requirements [22], [23]. BE applications, designed to enhance user experience, may function with

varying resource allocations depending on the available resources. Each can define their own isolation requirements.

While existing work has investigated dynamic container performance using automotive [16] or general [24] hardware, we have not found proposals for resource allocation strategies. In this work, we focus on studying the experimental validity of onboard resource allocation built on top of the previously mentioned state-of-the-art dynamic mechanisms.

III. METHODOLOGY

This section serves as a concise summary of the problem formulation presented in our previous paper [11]. We introduce the problem assumptions, goals, and modelling, followed by a brief overview of our proposed algorithm.

1) *Problem formulation:* Our approach starts with some assumptions and goals. We assume that each application is already assigned to a specific software context or ECU within the vehicle. The main goal of the algorithm is to continuously maximize the overall vehicle-wide user experience. We defined an algorithm that can make fast decisions in an embedded context, adjusting onboard functionalities when resource limitations hinder the allocation of the requested resources.

Let \mathcal{A} be the set of available applications in a vehicle app store. We separate \mathcal{A} in two partitions, namely \mathcal{A}_{SC} for safety-critical (SC) applications and \mathcal{A}_{BE} for best-effort (BE) applications, with $n_{SC} = |\mathcal{A}_{SC}|$ and $n_{BE} = |\mathcal{A}_{BE}|$. We assume that SC applications are not subject to the same resource constraints as BE applications, as their resources are guaranteed to be reserved by the vehicle architecture separately. This may be performed by methods such as [5] based on pre-allocations which are out of the scope of this work. In this setup, the vehicle may dynamically change the onboard resource allocations depending on the required SC features. Hence, resources for BE applications must adapt at runtime depending on the current context, such as V2X network saturation and available BE network bandwidth.

Each BE application $\mathcal{A}_i \in \mathcal{A}_{BE}$ defines a set of m_i runtime modes, denoted as \mathcal{A}_i^1 to $\mathcal{A}_i^{m_i}$, with varying levels of functionalities and resource requirements. \mathcal{A}_i^1 is the nominal mode with the most features and resource requirements, while $\mathcal{A}_i^{m_i}$ is the most degraded mode. Modes can have dependencies to other modes, forming a directed acyclic graph.

The vehicle can be modelled as a list of r resources $\mathcal{R}_1, \dots, \mathcal{R}_r$. The maximum hardware capacity of the vehicle is denoted as a vector of r positive reals R^{\max} such that $R^{\max}[i] \in \mathbb{R}^+$ denotes the maximal capacity of resource \mathcal{R}_i . Resources can describe the CPU and memory capacities for each ECU, available best-effort bandwidth for each physical network link in each direction, external system capacity such as V2X and edge computing availability, and more. However, resource availability can dynamically change at runtime. Hence, we define the current BE capacity of a vehicle as a vector R of r values such that $R[i] \in [0, R^{\max}[i]]$.

Each mode \mathcal{A}_i^j is associated with an AXIL rating $X_i^j \in \mathbb{R}^+$, expressing the perceived user experience for the application in this mode. The higher the AXIL rating, the better

TABLE I: AXIL definition. Just like ASIL, AXIL combines three parameters to assess the runtime priority of a service.

E_1	E_2	E_3			
		Minimal	Low	Medium	High
Easy	Rare	-	-	-	-
	Low	-	-	-	-
	Medium	-	-	-	A
	High	-	-	A	B
Medium	Rare	-	-	-	-
	Low	-	-	-	A
	Medium	-	-	A	B
	High	-	A	B	C
Difficult	Rare	-	-	-	A
	Low	-	-	A	B
	Medium	-	A	B	C
	High	A	B	C	D

Priority order: $- < A < B < C < D$

Legend →	E_1	E_2	E_3
ASIL	Controllability	Exposition	Severity
AXIL	Ease of Substitution		Quality of Exp.

the user experience. Figure 1 illustrates an example of randomly selected AXIL ratings for a set of applications. This creates a fictional but representative desired state instance. Additionally, each mode \mathcal{A}_i^j is associated with a resource requirements vector M_i^j , with $M_i^j[k] \in [0, R^{\max}[k]]$ expressing the resource requirement of resource \mathcal{R}_k to execute the mode. At any time, the vehicle may request any subset of \mathcal{A} .

At the core of the optimization problem is selecting the best combination of runtime modes for a given set of requested applications, with some applications potentially remaining disabled. The objective is to maximize overall user experience, defined as the sum of the AXIL scores of each active mode, while respecting resource limits and dependency relationships. As shown in our previous work, this translates into a mathematical problem comparable to the NP-Hard *knapsack* problem, requiring a heuristic solution.

2) *Proposed algorithm*: We proposed a heuristic solution for efficient runtime mode selection. In essence, the algorithm starts with all applications disabled and iteratively enables the most beneficial modes, starting from the most degraded mode upward, until the resource limits are reached. At each iteration, the next higher mode for each application is considered as a potential candidate to be upgraded. Then, one of the candidates is selected using a cost function, which divides the AXIL improvement by the resource cost of enabling each mode. The algorithm upgrades the selected mode and continues iterating until there is no improvement left. The algorithm also respects the dependencies between the modes, ensuring that a mode is only enabled if all its dependencies are also enabled. This approach is designed to be embedded onboard, making suboptimal but fast decisions in real-time. It can be re-executed when dynamic changes in resource availability and user requests occur to continuously adapt the onboard features.

While our previous simulation results demonstrated its effectiveness, an experimental study would greatly contribute in validating its industrial feasibility, notably using embedded

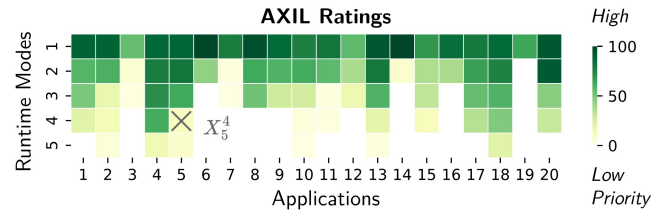


Fig. 1: Example of AXIL ratings attributed to the 1-5 runtime modes for each of the 20 applications. Higher modes provide more features, hence better QoE and execution priority.

hardware with capabilities resembling automotive ECUs to study transition delays and system reactivity.

IV. EXPERIMENTAL SETUP

This section presents the experimental setup used to evaluate our proposal. We describe our design choices for the test bench, the hardware used, the software architecture, the test scenarios, the data collection, and the evaluation metrics.

1) *Objectives*: In this study, we aim to provide empirical evidence of the benefits brought by our approach through experimental results. Hence, we focused on building a general, flexible, and representative platform rather than using specialized automotive hardware. This reduces development time and offers greater flexibility for implementations and scenarios, while still providing a realistic environment for the algorithm to be evaluated, as justified in the following paragraphs. Additionally, as mentioned in the related works, existing automotive platforms have already been used to investigate container performance [16] which is not our scope.

2) *Hardware architecture*: The test bench is designed to resemble a typical SDV architecture. It is composed of a set of 4 ECUs connected in a star topology using Ethernet links. The ECUs are represented using Raspberry Pi 4 Model B (RPi) single-board computers with 8Gb of RAM, connected through an Ethernet switch. As SDVs are expected to include mixed-criticality network traffic, all network interfaces are TSN-capable using an extension card for the RPi and TSN features in the switch model Relyum RELY-TSN4. See Figure 2 for a photo and simplified representation of the test bench. The RPi are connected to an external computer for monitoring through Wi-Fi, which enables network flow separation between scenario-relevant and monitoring traffic.

3) *Software architecture*: The software stack is designed to resemble a typical SDV software architecture. Each RPi runs a Linux-based operating system patched with a real-time kernel to enable TSN capabilities. The network stack supports the TSN standards including time synchronization (IEEE 802.1AS) and time-aware traffic shaping (IEEE 802.1Qbv). Finally, all ECUs and switch have an active local server supporting the NETCONF protocol, allowing for dynamic reconfiguration of the TSN ports. Hence, a centralized controller can dynamically change the TSN configuration of the network to adapt to the current vehicle context by calling the NETCONF server on each device with a precise time.

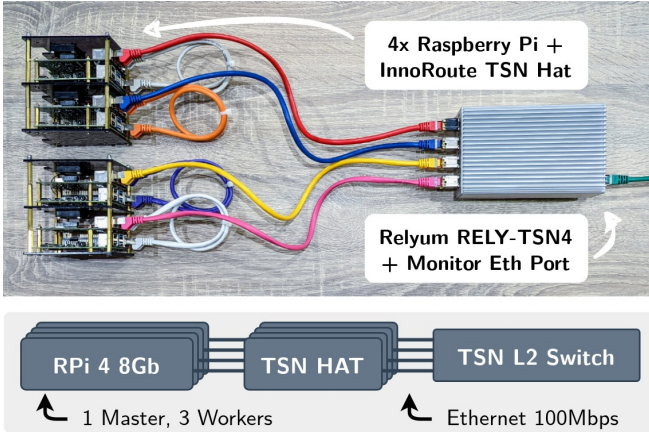


Fig. 2: Hardware architecture of test bench made of 4 ECUs connected in star topology using TSN-enabled network links.

Figure 3 shows a component view of the software stack. We use K3s as the Kubernetes distribution to orchestrate and distribute the applications and their runtime modes into the ECUs. This corresponds to the future trend of automotive software development for SDVs [25], which aims to facilitate software-hardware decoupling, continuous deployment, and dynamic resource allocation. Hence, the applications are containerized using Docker [24]. They are all launched from a common image, but they are configured with different runtime modes, network traffic generation profiles, and resource requirements from a manifest file. The applications are designed to generate best-effort network traffic and artificially consume resources following pre-determined requirements, such as CPU and memory usage. With this approach, we aim to mimic the typical resource constraints and behavior found in BE applications.

4) *Configuration generation*: Given a number of applications n and maximum number of runtime modes m_{\max} , we randomly generate a number of modes m_i for each application \mathcal{A}_i . We generate a random dependency graph $G_{\mathcal{A}}$ between applications with a target density by iteratively adding edges randomly to a graph up until the density is reached, and removing one edge per cycle if any appears. This lets us generate a second mode-level dependency graph $G_{\mathcal{M}}$. For each dependency edge in $G_{\mathcal{A}}$, we generate a random number of edges between modes of the two apps as long as they do not cross. Then, for each edge in $G_{\mathcal{M}}$, we generate a random number of network flows with random bandwidth requirements. Finally, each mode \mathcal{A}_i^j is attributed a random resource requirement $M_i^j[k]$ for each resource k , i.e. CPU and RAM in this work. Note that values are generated within the bounds set in Table II for fixed values. They also respect the structure defined in Section III along with its constraints.

We generate a manifest file for each application \mathcal{A}_i which describes the runtime modes, network flows, and resource requirements for each mode \mathcal{A}_i^j . When an application is launched in a specific mode, the common container image is started and configured with the corresponding manifest. The selected value ranges for each requirement have been

TABLE II: Problem parameter ranges for the test scenarios.

Resource	Value range
CPU usage	0-10%
Memory usage	0-200Mb
Number of modes	1-5
Number of flows per dependency	1-5
Network bandwidth requirement per flow	0.1-2Mbps

manually calibrated relative to the test bench capabilities, as the aim is to demonstrate the ability of the algorithm to adapt to the current platform. These ranges are shown in Table II.

Then, each mode is assigned a random AXIL rating in decreasing order, as illustrated in Figure 1 with a particular instance of 20 applications specifying at most 5 runtime modes each. Finally, each application is randomly assigned to one of the ECUs in the manifest, which we assume is provided by an external onboard scheduler out of our scope.

5) *Test scenario*: We define a simple yet representative test scenario to evaluate our algorithm. We aim to let the test bench run through a continuous change of vehicle states corresponding to subsets of active applications in the app store. To generate a state, a random set of applications is added and recursively extended with their dependencies. Then, we evaluate the test bench performance through continuous state changes, each with a random duration of 10 to 60 seconds.

As we have found the computing power of the RPis to be limited, the network bandwidth requirements are set to be relatively low. We set a 90% TAS closed duty cycle on all TSN ports, which effectively limits the physical bandwidth to 10Mbps. This setting can also be seen as if safety-critical applications have already reserved network resources for time-sensitive traffic. Therefore, the available bandwidth given to our optimization algorithm is 10Mbps per link to reflect the current available BE network state in the vehicle.

We repeat this scenario in two settings, (1) by activating all applications at their maximum runtime mode thereby bypassing the algorithm (baseline), and (2) by launching the applications at the runtime mode selected by the algorithm (optimized). This allows us to compare the utility of the algorithm compared to the lack of a resource-aware mechanism.

6) *Evaluation metrics*: We consider several evaluation metrics to assess the performance of the algorithm:

- **Network health**: As the main objective of our algorithm is to guarantee the allocation of resources specified by each application’s manifest, our primary metric aims to measure the overall system’s network health. We measure the network traffic generated and received by the applications. For each flow, we compare the observed network bandwidth with the expected generated bandwidth specified in its manifest file. This results in a time series of percentage values. We then aggregate the results with the median, Q1, and Q3 values as the indication of global vehicle health.
- **Resource usage**: We also measure the resource usage of the ECUs, namely CPU and memory usage, to assess

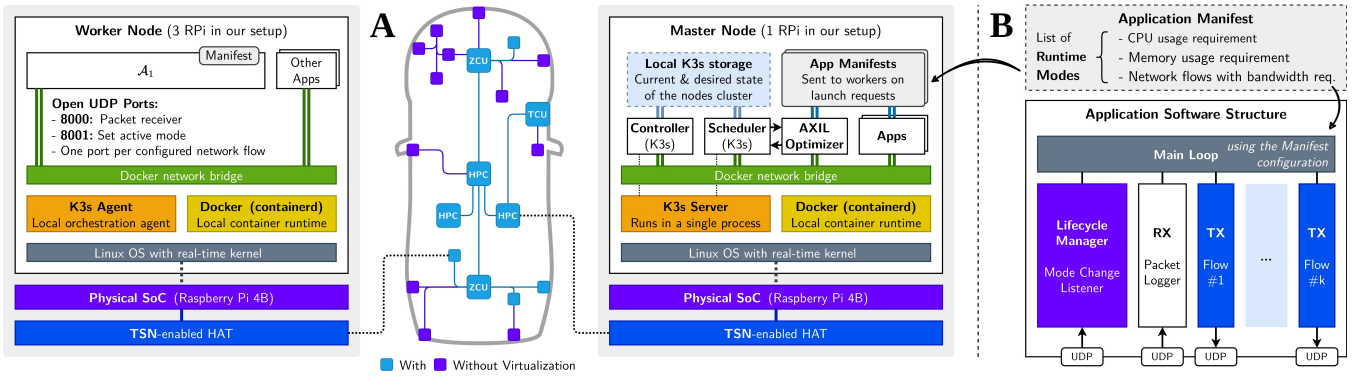


Fig. 3: Simplified representation of the software stack. (A) We use K3s as the Kubernetes distribution to orchestrate and distribute the containerized applications and their runtime modes into the ECUs. (B) The application is structured to generate mock best-effort network traffic according to a manifest file. These manifests are distributed to the ECUs at launch time.

the ability of the algorithm to respect the resource constraints of the vehicle. Observing a low network health along with saturated computing nodes would signal the necessity of resource-aware orchestration mechanisms.

- **Algorithm performance:** At each vehicle state change, we measure the time taken by our algorithm to generate a solution. Additionally, we also monitor the time to re-configure the K3s cluster once a new state is requested, and the time to adapt to dynamic changes in resource availability and user requests. This will provide insights into the performance of this architecture paradigm. Note that the total user experience is not directly measured in this study. We consider the algorithm always returns its best approximate solution as demonstrated in our previous work [11].

7) *Data collection:* While a scenario is running, the applications store the packets received, resource usages, and notable lifecycle events such as activation times and mode changes. They are then post-processed to extract the metrics.

V. RESULTS

The experimental results are presented in Figures 4 and 5. The results demonstrate that the baseline scenario shows severe stress in CPU usage, and the applications are not able to send network traffic at their target speeds. On the opposite, the optimized scenario chooses to launch the requested applications at degraded states to accommodate for the currently available resources, finding a suitable UX compromise. In this section, we start by studying the algorithm performance then analyze a full test scenario with random state changes.

1) *Adaptability in real-time:* On this hardware, the algorithm search times are shown in Figure 4. It is capable of producing solutions in 750ms to 2.6 seconds for approximately 30 applications with 1 to 4 modes each (size M). We believe this problem size to be a common use case in commercial vehicles, e.g. to manage user-focused or optional services.

This performance is suitable for occasional best-effort decision-making during vehicle operation, and can be re-executed when dynamic changes in resource availability

and user requests occur. However, limitations remain as the algorithmic complexity prohibits problem sizes larger than M in practice. Further optimizations can be achieved such as reimplementing the algorithm in a more efficient language (currently in Python), caching common results, pre-activating some modes depending on global state management rules, or performing calculations on the edge or cloud when available.

2) *Network performance:* Figure 5 shows the metrics collected from an experiment with an app store of size M. It compares the worst-case scenario where all applications are activated at their maximum runtime mode (Figure 5A) with the scenario where our algorithm is active (Figure 5B). The results show that the algorithm effectively respected the constraints of the vehicle as the network health score remains close to 100%. Hence, applications are degraded to their most efficient mode. The baseline shows an unpredictable and insufficient network performance due to over-provisioned resources, demonstrating the need of using resource-aware mechanisms. Instead, we maximize the UX-to-resource ratio.

Name	Nb. of Apps	Nb. of Modes	Dependencies
XS	10	1	5%
S	20	3	5%
M	30	4	10%
L	50	5	15%
XL	100	5	20%

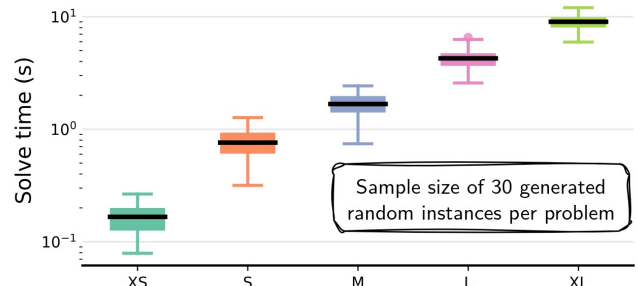


Fig. 4: Solving times on the test bench depending on the problem size with parameters given in the associated table.

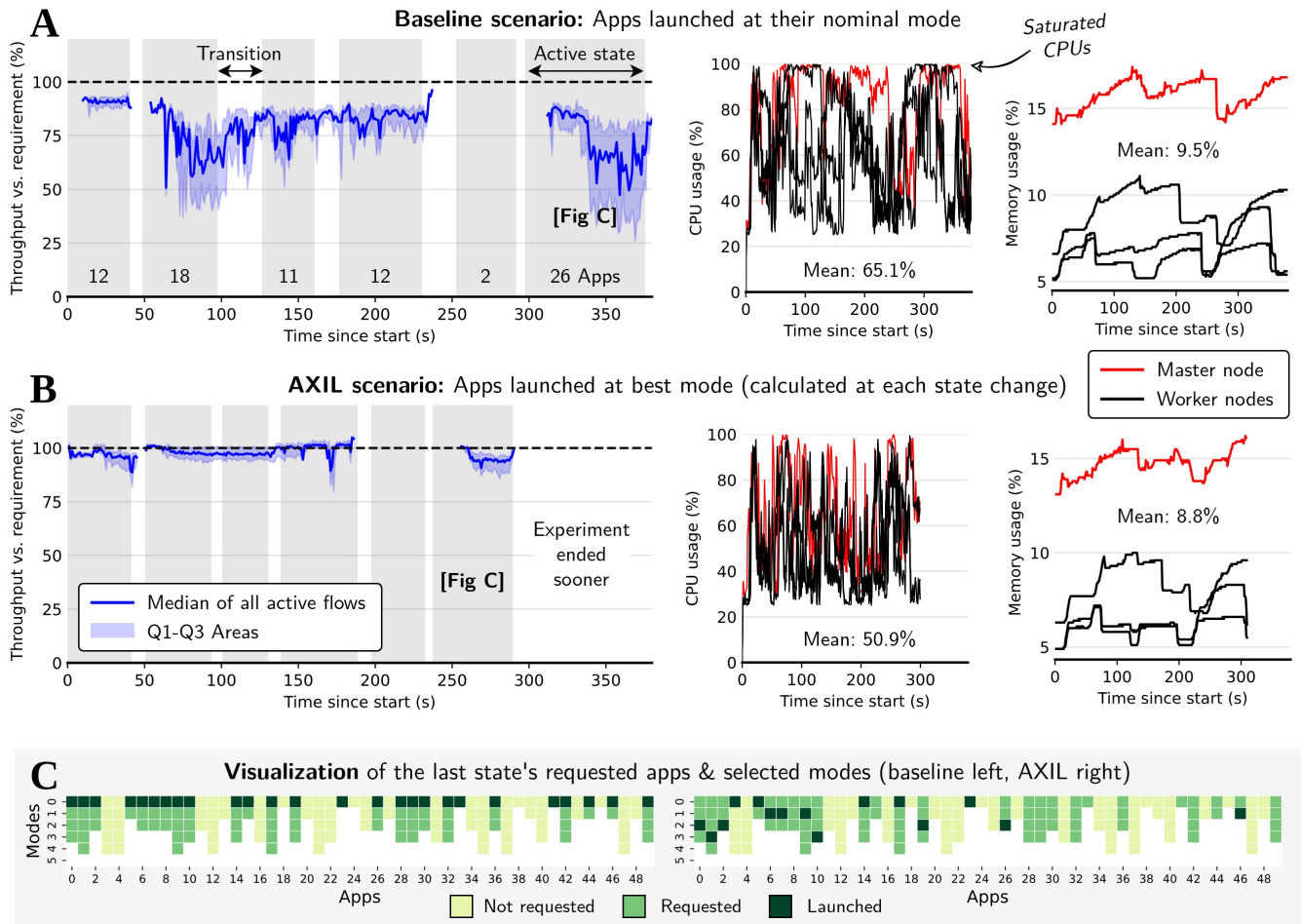


Fig. 5: Experimental results with an app store of 50 applications and 6 state changes of random sets of applications. We show our proposed network health metric when launching apps at (A) their nominal modes and (B) at the mode calculated by our algorithm. (C) Illustrates the runtime mode decisions in both scenarios with the requested applications and launched modes.

Therefore, only the optimized scenario with 100% network health permits the applications to obtain their required network resources specified in their manifests. Note that container launch delays explain temporary missing network statistics, and the 5th state includes 2 applications installed on the same node. Their traffic is not seen in our Ethernet-scoped statistics but is accounted for in the compute usage.

3) *Optimized resource usage*: The baseline scenario shows that the CPU usage of nodes regularly reaches saturation at 100%, also indicating over-provisioning. In the optimized case, only the master node rarely reaches CPU saturation mostly during transitions (i.e. scheduling, AXIL calculation, deployment). This behavior is desired in automotive systems to avoid performance bottlenecks and can be calibrated through the resource usage parameters. Memory usage is comparable in both scenarios, which could be due to the small memory footprints of the applications.

4) *Reduced transition times*: Figure 5 also shows a significant decrease in total transition times between vehicle states when the algorithm was applied. The transition times include stopping and launching the application containers, as well as the algorithm search time in the optimized case.

This is a direct result of using the available resources without overloading the system, which enables faster operations for K3s. The algorithm does not significantly impact the total transition times in this setup, as the search time is negligible compared to the container launch times.

The overall results lay a first stone for advancing dynamic onboard service orchestration in SDVs. However, we must note several limitations to our results. First, applications are implemented in Python, which introduces significant performance bottlenecks and may not reflect the maximal performance reachable with this hardware. This prohibited the study of packet latencies and jitter, as well as including time-sensitive network support for applications. Second, the current work does not dynamically change the available resources such as network capacity which limits the relevance of the results to real-world scenarios.

Limitations also apply to the proposed resource-aware dynamic methodology. First, the complexity of the algorithm prohibits large sets of requested applications. The computation time remains too high for time-critical decisions, and the algorithm has no real-time guarantees. This may limit use cases as many onboard operations must be performed within

strict global timing requirements. Second, this dynamic management approach is only valid for microprocessor-based environments, as it requires K3s and Docker. Finally, the algorithm is limited by the current BE resources available on the vehicle and has no control over allocations for other domains such as SC apps. However, it presents an opportunity to investigate the deployment of more sophisticated SC applications within complex ECUs, such as High Performance Computer (HPC) ECUs, where an increasing number of safety, body, and infotainment applications are being allocated using shared resources.

VI. CONCLUSION

This paper introduces an experimental investigation into a heuristic algorithm designed for efficient resource-based dynamic application orchestration in SDVs. The algorithm prioritizes user experience by optimizing the selection of runtime modes of requested applications within resource constraints and dependency relationships set by developers.

Conducted on a dedicated test bench mimicking a typical SDV architecture, the study employs Raspberry Pi single-board computers connected via Ethernet supporting TSN standards. Results illustrate the algorithm's capability to continuously adapt onboard functionalities while keeping the resource usage within ECU and network capacities. The concept of considering runtime modes combined with this algorithm guarantees sane network health and resource usage, independently of the ever-changing user requests. It also decreases transition times between vehicle states, compared to launching all apps without using any resource control mechanism. In this setup, the optimization algorithm can perform decisions in approximately one second for 20 applications with 1-5 modes each, which is a promising result for continuous orchestration. Room for performance improvement remains if the presented system is reimplemented using embedded automotive-grade technologies.

These results lay a foundation for advancing dynamic onboard service orchestration in SDVs. Future work will focus on extending the study to more intricate scenarios and diverse applications, as well as including other application domains such as safety-critical and cooperative V2X services.

ACKNOWLEDGMENTS

This work was supported by Stellantis under the collaborative CIFRE framework UTC/CNRS/PCA (ANRT contract n°2021/0865) with Heudiasyc. The authors would like to thank Charles Perold for his technical contributions.

REFERENCES

- [1] C. Buckl, A. Camek, G. Kainz, C. Simon, L. Mercep, H. Stähle, and Knoll, "The Software Car: Building ICT Architectures for Future Electric Vehicles," in *2012 IEEE International Electric Vehicle Conference*.
- [2] H. Askaripoor, M. Hashemi Farzaneh, and A. Knoll, "E/E Architecture Synthesis: Challenges and Technologies," *Electronics*, 2022.
- [3] S. Jiang, "Vehicle E/E Architecture and Key Technologies Enabling Software-Defined Vehicle," SAE International, Warrendale, PA, SAE Technical Paper 2024-01-2035, 2024, iSSN: 0148-7191, 2688-3627.
- [4] N. Ayres, L. Deka, and D. Paluszczyszyn, "Continuous Automotive Software Updates through Container Image Layers," *Electronics*, 2021, publisher: Multidisciplinary Digital Publishing Institute.
- [5] P. Laclau, S. Bonnet, B. Ducourthial, X. Li, and T. Lin, "Predictive Network Configuration with Hierarchical Spectral Clustering for Software Defined Vehicles," in *2023 IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*, 2023, pp. 1–5, iSSN: 2577-2465.
- [6] M. Schindewolf, H. Stoll, H. Guissouma, A. Puder, E. Sax, A. Vetter, M. Rumez, and J. Henle, "A Comparison of Architecture Paradigms for Dynamic Reconfigurable Automotive Networks," in *2022 International Conference on Connected Vehicle and Expo (ICCVE)*.
- [7] M. Haerberle, F. Heimgaertner, H. Loehr, N. Nayak, D. Grewe, S. Schildt, and M. Menth, "Softwarization of Automotive E/E Architectures: A Software-Defined Networking Approach," in *2020 IEEE Vehicular Networking Conference (VNC)*, 2020.
- [8] V. Bandur, G. Selim, V. Pantelic, and M. Lawford, "Making the Case for Centralized Automotive E/E Architectures," *IEEE Transactions on Vehicular Technology*, vol. PP, pp. 1–1, 2021.
- [9] T. Häckel, P. Meyer, F. Korf, and T. C. Schmidt, "Secure Time-Sensitive Software-Defined Networking in Vehicles," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 1, pp. 35–51, 2022.
- [10] K. Taylor, "Digital cockpit in the era of the software-defined vehicle," SAE International, Warrendale, PA, SAE Technical Paper 2024-01-2391, 2024, iSSN: 0148-7191, 2688-3627.
- [11] P. Laclau, S. Bonnet, B. Ducourthial, X. Li, and T. Lin, "Enhancing Automotive User Experience with Dynamic Service Orchestration for Software Defined Vehicles," 2024, preprint, submitted to IEEE ITS, under revision. [Online]. Available: <https://hal.science/hal-04505345>
- [12] A. Frigerio, B. Vermeulen, and K. Goossens, "Component-Level ASIL Decomposition for Automotive Architectures," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2019, pp. 62–69.
- [13] S. Kugele, P. Obergfell, M. Broy, O. Creighton, M. Traub, and W. Hopfensitz, "On Service-Oriented for Automotive Software," in *2017 IEEE International Conference on Software Architecture (ICSA)*.
- [14] J. Frtunikj, M. Armbruster, and A. Knoll, "Run-Time Adaptive Error and State Management for Open Automotive Systems," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*.
- [15] X. Liao, Z. Wang, X. Zhao, K. Han, P. Tiwari, M. J. Barth, and G. Wu, "Cooperative Ramp Merging Design and Field Implementation: A Digital Twin Approach Based on Vehicle-to-Cloud Communication," *IEEE Transactions on Intelligent Transportation Systems*, 2022.
- [16] T. Häckel, P. Meyer, L. Stahlbock, F. Langer, S. A. Eckhardt, F. Korf, and T. C. Schmidt, "A Multilayered Security Infrastructure for Connected Vehicles – First Lessons from the Field," 2023, arXiv:2310.10336 [cs].
- [17] B. Shi, X. Tu, B. Wu, and Y. Peng, "Recent Advances in Time-Sensitive Network Configuration Management: A Literature Review," *Journal of Sensor and Actuator Networks*, vol. 12, no. 4, p. 52, 2023.
- [18] K. Halba, C. Mahmoudi, and E. Griffor, "Robust Safety for Autonomous Vehicles through Reconfigurable Networking," *Electronic Proceedings in Theoretical Computer Science*, vol. 269, 2018.
- [19] T. Hackel, P. Meyer, F. Korf, and T. C. Schmidt, "Software-Defined Networks Supporting Time-Sensitive In-Vehicular Communication," in *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019, iSSN: 2577-2465.
- [20] A. Kampmann, B. Alrifaae, M. Kohout, A. Wustenberg, T. Wopen, M. Nolte, L. Eckstein, and S. Kowalewski, "A Dynamic Service-Oriented Software Architecture for Highly Automated Vehicles," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. Auckland, New Zealand: IEEE, 2019, pp. 2101–2108.
- [21] E. Truyen, D. Van Landuyt, D. Preuveneers, B. Lagaisse, and W. Joosen, "A Comprehensive Feature Comparison Study of Open-Source Container Orchestration Frameworks," *Applied Sciences*, 2019.
- [22] Y. Peng, B. Shi, T. Jiang, X. Tu, D. Xu, and K. Hua, "A Survey on In-Vehicle Time-Sensitive Networking," *IEEE Internet of Things Journal*, vol. 10, no. 16, pp. 14 375–14 396, 2023.
- [23] D. F. Külzer, S. Stańczak, and M. Botsov, "Novel QoS Control Framework for Automotive Safety-Related and Infotainment Services," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*, 2020, pp. 1–7, iSSN: 1558-2612.
- [24] D. Fernández Blanco, F. Le Mouël, T. Lin, and A. Rekik, "Can Software Containerisation Fit the Car On-Board Systems ?" 2023. [Online]. Available: <https://hal.science/hal-04127629>
- [25] N. Nayak, D. Grewe, and S. Schildt, "Automotive Container Orchestration: Requirements, Challenges and Open Directions," in *2023 IEEE Vehicular Networking Conference (VNC)*, 2023, pp. 61–64.