Predictive Network Configuration with Hierarchical Spectral Clustering for Software Defined Vehicles

Pierre Laclau[1,2], Stéphane Bonnet[1], Bertrand Ducourthial[1], Xiaoting Li[2] and Trista Lin[2]
[1]Heudiasyc, UMR CNRS, Université de Technologie de Compiègne, France
[2]Stellantis, Vélizy-Villacoublay, France
{pierre.laclau, stephane.bonnet, bertrand.ducourthial}@utc.fr, {xiaoting.li, trista.lin}@stellantis.com

***Abstract*—The increasing connectivity and autonomy of vehicles has led to a growing need for dynamic and real-time adjustments to software and network configurations. Software Defined Vehicles (SDV) have emerged as a potential solution to adapt to changing user needs with continuous updates and onboard reconfigurations to offer infotainment, connected, and background services such as cooperative driving. However, network configuration management in SDVs remains a significant challenge, particularly in the context of shared Ethernet-based in-vehicle networks. Traditional worst-case static configuration methods cannot efficiently allocate network resources while ensuring Quality of Service (QoS) guarantees for each network flow within the physical topology capabilities. In this work, we propose a configuration generation methodology that addresses these limitations by dynamically switching between pre-computed offboard configurations downloaded to the vehicle. Simulation results are presented and future work is discussed.**

***Index Terms*—Software Defined Vehicle (SDV), Dynamic network configuration, Service Oriented Architecture (SOA), Time Sensitive Networking (TSN), In-vehicle networks (IVN).**

I. Introduction

In the last few years, automotive OEMs have started to develop a range of complex features such as dedicated infotainment app stores, automated driving, and Vehicle-to-Everything (V2X) services. This shift invites OEMs to continuously deliver these features through seamless Over-the-Air (OTA) updates, which contributes to longer lasting vehicles in the hope to bring the industry closer to sustainable transportation systems. As a result, future vehicles may resemble '*Smartphones on Wheels*' where users can dynamically request services throughout the vehicle lifetime while OEMs continuously integrate background services such as autonomous and cooperative driving, smart grid algorithms, and more [1], [2].

To support this paradigm shift, the automotive industry is undergoing a rapid transformation toward Software Defined Vehicles (SDV) [3]. Previously, Electric and Electronic (E/E) architectures were hardware-defined by integrating many single-function Electronic Control Units (ECU) into domain-centric networks. However, the increasing number of ECUs as well as more demanding network and OTA requirements are reaching the limits of current architectures [4]. These changes are motivating a global shift towards Zonal Oriented Architectures (ZOA), where fewer High Performance Computing (HPC) ECUs are expected to group and manage multiple heterogeneous functions. ZOA allows for a centralized and therefore cost-effective reservation of computational resources for future updates and features. In addition, Ethernet introduces service-oriented communications and increased bandwidth [5].

This new hardware approach must be controlled by a software stack capable of applying OTA updates and dynamically switching the onboard software context to provide relevant applications. A common solution is to deploy a Service Oriented Architecture (SOA) which handles high-level communications through publish-subscribe protocols and the orchestration of services based on user requests, vehicle context, and available updates, allowing for dynamic reconfigurations of software and network resources [6]. However, SOA does not address the guarantee of Quality of Service (QoS) constraints such as hard real-time latency and jitter for safety-critical flows. Hence, the onboard architecture also relies on an embedded infrastructure based on technologies such as Software Defined Networking (SDN) and Time Sensitive Networking (TSN) that can be used to handle dynamic routing and hard real-time scheduling of network resources on a per-flow basis.

Such an architecture depends on *configurations* to describe the behavior of physical and virtual network components. These configurations determine how traffic shapers, priorities, buffers, routes, firewalls, sleep modes, and others are allocated to meet all mixed QoS constraints. However, generating these configurations is a complex and often NP-hard task [7] as it requires generating a coherent global configuration across all network components despite their heterogenous implementations and parameters. For instance, some components such as 802.1Qbv for TSN rely on SAT-based solvers to generate valid configurations (e.g. TSNsched [8]). It would thus be infeasible to generate configurations onboard due to limited resources.

We believe that current methods for determining network configurations using worst-case static approaches are no longer sufficient for SDVs due to the potential presence of thousands of diverse applications and the need for frequent updates. Therefore, it is necessary to find new approaches to manage the dynamic nature of network configurations. In this work, we propose a novel methodology for generating these configurations, which dynamically reconfigures the in-vehicle network using pre-computed offboard configurations downloaded to the vehicle. We aim to improve the flexibility and efficiency of real time embedded networks while minimizing onboard resources.

This work was supported by Stellantis under the collaborative framework UTeam/UTC/CNRS/PCA with Heudiasyc (ANRT contract n°2021/0865).

II. Related work

Traditionally, network configurations have been determined using static worst-case approaches, where the network is configured to handle the worst possible scenario in terms of traffic and QoS requirements for the entire vehicle lifetime [9], [10]. There has been a considerable amount of research conducted on the generation of network configurations that varies based on the technologies and components used, such as statistical checks to validate CAN networks signal timing ranges, TSN configurators for various traffic shapers [8], or SDN orchestrators with dynamic routing engines [11].

However, these methods assume that the input set of flows will produce a feasible configuration. Nonetheless, with the rise of SDVs and the potential for thousands of applications with diverse flow requirements, the traditional static approach becomes inadequate as it may not be able to allocate all flows into the physical in-vehicle bandwidth-limited links [12]. Furthermore, with a continuously evolving offering of applications in the app store and ever-improving background services, network configurations will need to be updated regularly.

On the other hand, traditional reconfiguration methods used to apply new configurations are also limited. These methods rely on infrequent updates through OTA updates and often require reflashing ECUs and network components. This process takes time [13] and consumes onboard energy resources, making it impractical to make real-time adjustments to flow allocations. As a result, these traditional methods fail to meet the dynamic requirements of SDVs. To address this limitation, new approaches that can handle the complexity and dynamic nature of network configurations are currently being developed. For instance, research initiatives by CoRE-RG seek to provide network reconfiguration capability at runtime for maximum orchestration flexibility [14]. However, these methods only apply configurations produced by other components.

We believe that the unique challenges posed by an environment with thousands of available services are not addressed by existing research. Our work aims to bridge this gap by leveraging existing network schedulers and dynamic reconfiguration frameworks to support (1) a large number of applications, (2) dynamic reconfigurations, and (3) minimal onboard overhead.

III. Problem formulation

We propose a new configuration generation methodology that allows for dynamic switching between pre-computed offboard and pre-downloaded onboard configurations. Our approach is based on the assumption that a vehicle will not require all services to be active simultaneously. We believe this to be a reasonable assumption in the context of app stores, as a large part of the available services (such as infotainment and cooperative V2X services) are functionally mutually exclusive depending on the vehicle and user contexts. We can thus generate smaller configurations that consider only subsets of applications that may be enabled simultaneously, thus extending the physical capabilities of the vehicle.

Let $A = \{a_1, ..., a_n\}$ be the set of applications available in the app store, including background services, along with

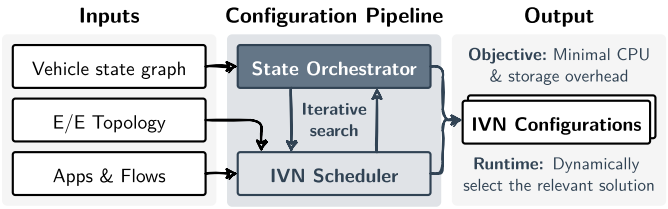

Fig. 1. Block diagram of the configuration methodology. Our goal is to orchestrate the selection of network flows to produce a set of feasible configurations, using an off-the-shelf in-vehicle network scheduler for all available applications within the onboard E/E physical capacities.

$S = \{s_1, ..., s_m\}$ a finite set of states where each state $s_i \subset A$ is a set of applications that could be activated concurrently. Each state may require a residual free bandwidth for unallocated *best-effort* streams in advance, and it is assumed that all application dependencies are respected. For each pair of states s_i and s_j, let $\mathbb{P}(i, j)$ be the probability of transition from state i to state j and $P_S = \{\mathbb{P}(i, j),\ s_i, s_j \in S\}$ the transition probability matrix linking all states of S. Therefore, we can define the vehicle state graph $G = \{S, P_S\}$, illustrated with an example in Figure 2A, as a strongly connected Markov chain which represents the different combinations of active applications that the vehicle may encounter at runtime. The state of the vehicle thus evolves as a random-walk on G.

The problem is to find the set of feasible configurations C, each generated by an architecture-dependent global network scheduler, that meets industry-relevant priorities defined below.

First, we minimize computing overhead by reducing the frequency of transitions between configurations. This optimizes resources dedicated to functional features, reduces hardware costs and energy usage, and minimizes user wait times when using traditional re-flashing reconfiguration methods (e.g. for legacy ECUs). Secondly, we mitigate the amount of data transmitted and stored onboard by minimizing the number of configurations that need to be generated. While the car will likely often have WiFi available while parking, this may not be true all the time which would require paid mobile data usage.

Note that these two objectives are only sufficient to guarantee safe reconfigurations while parked, meaning no active flows remain active during transitions. While driving, some packets from previously active applications might still be in transit inside the network, which would potentially lead to lost packets as configurations are generated independently with separated buffer usage estimates. In this work, we focus on orchestrating the generation of independent configurations by building on top of existing network schedulers for parked reconfigurations, as current schedulers do not support the generation of multiple coherent configurations.

Finally, we minimize the number of calls to the network scheduler as it might take a considerable amount of time to check the validity of a particular set of flows. The configurations depend on the E/E physical topology which includes the set of ECUs available in the vehicle E, as well as the list of all potential flows F required by applications as inputs for

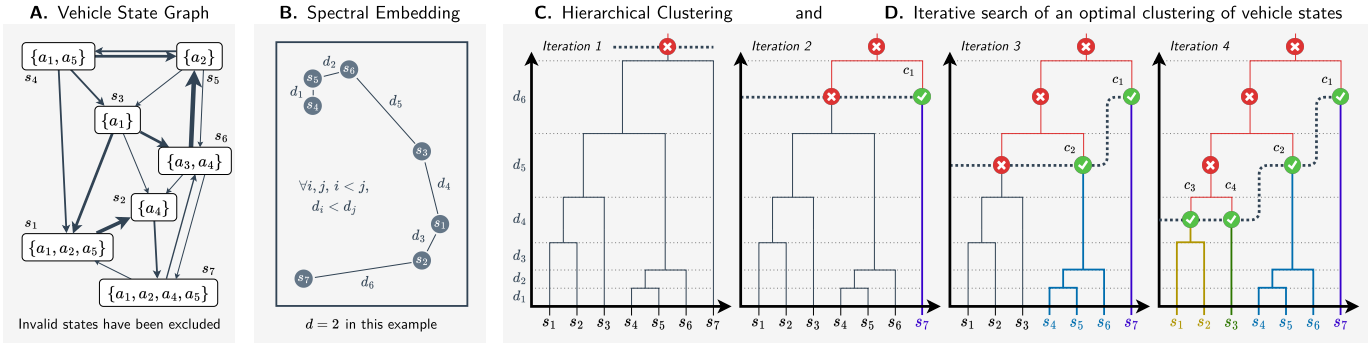

Fig. 2. Illustration of our predictive configuration methodology. **(A)** represents a vehicle state graph G modelled as Markov chain with nodes defined as sets of active applications and weighted edges representing reconfiguration probabilities. **(B)** shows the 2-dimensional spectral embedding of G, and **(C)** points to the generated tree structure called dendrogram using hierarchical clustering. **(D)** shows our iterative search algorithm, which explores the tree from the root downwards and calls the network scheduler at each branch to get its validity (green/red icons) until the final set of configurations $C = \{c_1, ..., c_k\}$ is found.

the network scheduler. Hence, each application is defined as $a_i = \{e, f\}$ with $e \in E$ the predetermined host ECU for its execution and $f \subset F$ the set of QoS-dependent flows required. This serves as the input data for the global network scheduler. The diversity of parameters and flow types can vary depending on the capabilities of the selected scheduler, which can be considered as as a *black box* in this work.

In an industrial context, the vehicle state graph G can be generated based on expert system modelling, vehicle simulation, and real-world observation feedback. However, we assume that G might contain thousands of states, some of which could be subsets of other states. Hence, our objective is to process this input graph such that the final set of precomputed configurations remains minimal. Therefore, finding C can be done by solving the following optimization problem:

$$\underset{C}{\operatorname{argmin}}\, g(C) \quad \text{with} \quad g(C) = k + W + h \tag{1}$$

$$\text{such as} \quad k = |C| \quad \text{and} \quad W = \sum_{s_i, s_j \in C} P_C(i, j)$$

where C is the final set of generated configurations, k is the number of configurations, W is the sum of transition probabilities between configurations with P_C the transition matrix constructed from the subgraph of G clustered using C, and h is the number of calls to the global network scheduler.

IV. Methodology

The purpose of the *State orchestrator*, illustrated in Figure 1, is to orchestrate the selection of flow sets to achieve the previous optimization goal. Our methodology, illustrated in Figure 2, consists of the following steps.

Initialization (Figure 2A). First, it is possible to filter G by running the scheduler on each state to remove unfeasible states. Similarly, it is possible to filter out states already included in others based on the remaining bandwidth and energy consumption goals of the vehicle.

Projection (Figure 2B). Next, we apply the *spectral embedding* method [15] where d is the destination dimension to be calibrated in the industrialization phase such that $2 \leq d \ll |S|$. Thus, the relative coordinates of each state in the embedded space correspond to their probabilistic relations. This has the effect of moving states with high transition frequencies closer together. Since G is directed, we use the asymmetric Laplacian matrix as the similarity matrix between nodes to account for irregular probabilities [16].

Hierarchical Clustering (Figure 2C). We then apply the *agglomerative hierarchical clustering* method [15] using the embedded space by merging the states in pairs following the increasing order of Euclidean distance (Y axis in Figure 2C) between states, which groups states with high transition probabilities first. This results in a tree structure according to their probabilistic relationships. At this point, we obtain a *dendrogram* where each branch represents a candidate cluster of states. Each branch can be seen as a new state coming from the merging of the applications contained in the children states.

Each time two states s_i and s_j are clustered, the following occurs: first, the number of clusters k decreases by 1, which invites us to select the highest branches in the tree. Second, the transitions $P_{S\,ij}$ or $P_{S\,ji}$ become null as both states are merged into $s_i \cup s_j$ and the vehicle does not need to reconfigure its network as long as the active set of applications remains within this new set. Finally, it is necessary to execute the scheduler to know the validity of each branch.

Iterative search (Figure 2D). It follows from the three preceding remarks that $g(C)$ can only increase as we explore the dendrogram starting at the root downward by running the scheduler at each branch encountered. The algorithm thus consists in finding the branches of maximum height in the dendrogram producing valid configurations by exploring the topology from top to bottom. Since the algorithm stops at the first combination of valid branches found and $g(C)$ increases during the exploration, the solution corresponds to the minimum of g while constrained to return a set of valid configurations. In the end, all states s_i obtain a valid configuration generated at one of the parent branches.

The algorithm starts with the root of the topology c_r which corresponds to the ideal solution that minimizes all objectives. It is also equivalent to the worst case static scheduling methods

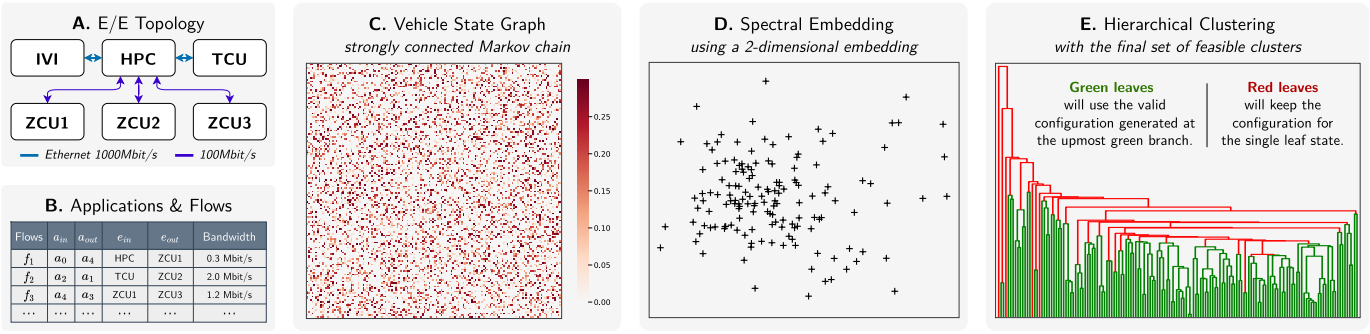

Fig. 3. Simulation results. We generate a simple scenario with **(A)** a zonal-oriented E/E architecture topology, **(B)** a random set of flows with parameters supported by the scheduler model, and **(C)** a heatmap that represents the P_S transition matrix of a random strongly connected Markov chain G. **(D)** shows the 2-dimensional spectral embedding of G, and **(E)** presents the generated dendrogram with the final set of iteratively searched configuration clusters.

found in the state of the art. We run the network scheduler on this single merged state and return $C = \{c_r\}$ if the scheduler generates a valid configuration. Otherwise, we repeat the procedure independently for all children of the node until the first valid branches are found. This later case is more probable in the context of thousands of available applications. Thus, we obtain $C = \{c_1, ..., c_k\}$ solution of the optimization problem posed in Equation 1.

The overall algorithm is of polynomial complexity. The complexity of the setup stage (corresponding to steps A, B, and C in Figure 2) is $\mathcal{O}(n^2)$ with $n = |S|$ (e.g. see [15]). This phase only needs to be performed once with a given set of applications. Then, the iterative search (Figure 2D) is reduced to $\mathcal{O}(n)$ since at most the $2n - 1$ nodes of the cluster tree topology need to be explored, which allows for an efficient generation of configuration sets for multiple physical vehicle topologies using a single dendrogram. Note that each scheduler call for a candidate branch c_i has a complexity that varies with the chosen model, such as $\mathcal{O}(|c_i|^2)$ for TSNsched [8].

V. Simulation Setup

To test the performance of our approach, we designed a simulation setup with a simple but representative scheduler model using randomly generated inputs. First, the E/E physical topology shown in Figure 3A follows a standard zonal-oriented star topology connected with Ethernet links of different datarates. Then, we generate a random dependency graph of $n = 500$ applications to represent an app store as a binomial directed graph. Each application is randomly assigned to one of the ECUs, which we assume is provided by an external onboard service orchestrator.

We selected a simple scheduler model that considers a unique *bandwidth* parameter and checks if the sum of the active flows in each link does not exceed its physical capacity. Then, we generate a random number of 1 to 5 flows for each edge in the dependency graph, with each being assigned a randomized bandwidth requirement between 0.1 and 5 Mbit/s (see Figure 3B). Additional constraints such as latency and jitter could be added when using a more complex scheduler.

These inputs are sufficient to run traditional worst case methods. To compare these methods with our dynamic approach, we also generate the random strongly connected Markov chain G shown in Figure 3C. This is done by taking the largest strongly connected subgraph of a random directed graph with $m = 500$, which produces a graph of $m = 433$ states. We then produce a random combination of active applications such that $s \subseteq A$ by sampling a random set of 5 to 500 applications and adding their dependencies. We also replaced existing states until the union set of all states contained all applications in A to produce a coherent input set.

Then, we filter G by removing states with unfeasible configurations for this particular E/E topology. Note that these invalid states may become valid for higher-end topologies. This methodology produces a final graph G of size $m = 150$. Edges are valued using random transition weights. Our algorithm does not assume any similarity between the sets of active apps between neighbor states in G, since its goal is to minimize the frequency of reconfigurations. Finally, we define r_A as the ratio of applications in A that have been allocated in at least one valid configuration for each scenario.

We study three simulation scenarios. Firstly, we attempt to schedule all flows at once by calling the scheduler once, which is equivalent to the worst case strategies (*worst-case* scenario). If the scheduler does not return a valid configuration, we sample random combinations of applications using the same methodology to generate S in order to estimate the performance of this strategy in an app store environment. Secondly, we generate one configuration for each state s_i in the state graph (*unfiltered* scenario). Lastly, we run our clustering strategy to reduce the number of final states (*reduced* scenario).

To evaluate the performance of each scenario, we compare each part of the objective function g, namely the number of produced configurations k, the sum of transition probabilities between states W, and the total number of scheduler calls h.

VI. Results

The simulation results are summarized in Table I. They indicate that the worst-case static approach is insufficient for allocating all flows in a single configuration. When sampling 1000 random feasible application combinations, the maximum amount of apps that can be allocated to a single configuration

TABLE I
EVALUATION METRICS FOR THE SIMULATION RESULTS

	Worst case c_r	Unfiltered S	Reduced C	Gains $S \rightarrow C$
r_A	13% (mean)	100%	100%	0%
k	1	150	39	74%
W	0	3411.5	413.7	87.9%
h	1	150	57	62%

using the worst-case approach was found to be only 28% with a mean of 13%. Even though $g(c_r)$ is minimal, this implies that the worst-case approach cannot guarantee an efficient allocation of all flows in a single configuration.

On the other hand, generating one configuration for each state $s \in S$ in the initial state graph resulted in 100% of valid configurations as expected by design. However, this approach generated a large number of configurations $k = 150$, which makes it impractical to be used in real-world scenarios due to the increased complexity and storage requirements. This solution also requires frequent onboard reconfigurations.

The proposed approach achieved better performance, generating 39 configurations while maintaining 100% valid configurations with transition probabilities reduced by 87.9% and scheduler calls by 62%. This suggests that our approach is efficient in allocating all flows with valid configurations, which makes it more practical for real-world deployment.

Our results demonstrate the effectiveness of our proposed methodology in reducing the number of configurations required while achieving a 100% success rate in valid configurations. This approach can be used in real-world scenarios where the number of ECUs and applications can be much larger, and hence the proposed methodology can provide an effective solution for dynamic configuration allocation.

Our results highlight the importance of a dynamic approach to configuration allocation, and the proposed approach can help improve the efficiency and scalability of automotive networks while ensuring a high level of QoS for all flows.

VII. CONCLUSION

We have presented a methodology for generating a set of offline network configurations to cover the dynamic use cases of vehicles. The configurations are generated with the aim to minimize the frequency of reconfigurations in the vehicle and storage requirements while reducing the number of network scheduler calls. Finally, the vehicle dynamically selects the correct pre-downloaded configuration on context changes.

This approach applies classical clustering strategies in an industrial environment to enable a large number of applications despite limited onboard hardware capabilities. Additionally, it is possible to optimize the offboard execution times in an industrial context by starting the exploration at an intermediate level in the dendrogram or by skipping multiple descendants at once, depending on calibration. Furthermore, the branches are independent and can therefore be evaluated in parallel.

This scenario can be particularly useful in the context of shared vehicles, where the vehicle reconfigures itself before a user reservation period based on their preferences, profile, and subscriptions. Moreover, the same state graph G can be used for several vehicle models with different topologies. However, one major limitation remains: this algorithm is independent from the network scheduler, which makes it impossible to construct multiple coherent configurations based on QoS guarantees to ensure safe reconfigurations on running vehicles.

Future work will aim to (1) test the network performance of our solution in a realistic automotive environment using more complex schedulers such as TSNsched [8] along with realistic input applications and flows, (2) improve current schedulers to consider transition safety and enable reconfigurations while driving, (3) extend the current architecture with an onboard configuration generator based on heuristics for non-critical flows, and (4) study the impacts of different configuration orchestration strategies on onboard energy consumption.

REFERENCES

[1] M. Haeberle, 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)*, Dec. 2020, pp. 1–8.

[2] X. Hu, K. Wang, X. Liu, Y. Sun, P. Li, and S. Guo, "Energy management for EV charging in software-defined green vehicle-to-grid network," *IEEE Communications Magazine*, vol. 56, no. 5, pp. 156–163, 2018.

[3] 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*, 2012.

[4] 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, Jan. 2021.

[5] H. Kim, M. Choi, M. Kim, and S. Lee, "Development of an Ethernet-Based Heuristic Time-Sensitive Networking Scheduling Algorithm for Real-Time In-Vehicle Data Transmission," *Electronics*, 2021.

[6] M. Rumez, D. Grimm, R. Kriesten, and E. Sax, "An Overview of Automotive Service-Oriented Architectures and Implications for Security Countermeasures," *IEEE Access*, vol. 8, pp. 221 852–221 870, 2020.

[7] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. Elbakoury, "Performance Comparison of IEEE 802.1 TSN Time Aware Shaper (TAS) and Asynchronous Traffic Shaper (ATS)," *IEEE Access*, vol. 7, pp. 44 165–44 181, 2019.

[8] A. C. T. dos Santos, B. Schneider, and V. Nigam, "TSNSCHED: Automated Schedule Generation for Time Sensitive Networking," in *2019 Formal Methods in Computer Aided Design (FMCAD)*, 2019.

[9] S. Quinton, T. T. Bone, J. Hennig, M. Neukirchner, M. Negrean, and R. Ernst, "Typical worst case response-time analysis and its use in automotive network design," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, Jun. 2014, pp. 1–6.

[10] J.-d. Park, B.-m. Cheoun, and J.-w. Jeon, "Worst-case analysis of ethernet AVB in automotive system," in *2015 IEEE International Conference on Information and Automation*, Aug. 2015, pp. 1696–1699.

[11] K. Halba, C. Mahmoudi, and E. Griffor, "Robust Safety for Autonomous Vehicles through Reconfigurable Networking," *Electronic Proceedings in Theoretical Computer Science*, vol. 269, pp. 48–58, Apr. 2018.

[12] Z. Zhou, J. Lee, M. S. Berger, S. Park, and Y. Yan, "Simulating TSN traffic scheduling and shaping for future automotive Ethernet," *Journal of Communications and Networks*, vol. 23, no. 1, pp. 53–62, Feb. 2021.

[13] N. Ayres, L. Deka, and D. Paluszczyszyn, "Continuous Automotive Software Updates through Container Image Layers," *Electronics*, 2021.

[14] 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.

[15] R. J. Sánchez-García, M. Fennelly, S. Norris, N. Wright, G. Niblo, J. Brodzki, and J. W. Bialek, "Hierarchical Spectral Clustering of Power Grids," *IEEE Transactions on Power Systems*, 2014.

[16] Q. Zheng and D. B. Skillicorn, "Spectral Embedding of Directed Networks," in *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*. New York, NY, USA: Association for Computing Machinery, 2015.