



UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
DEPARTMENT OF INDUSTRIAL ENGINEERING
FONDAZIONE BRUNO KESSLER

Doctorate Program in Industrial Innovation

DESIGN SPACE EXPLORATION FOR
ELECTRICAL/ELECTRONIC
ARCHITECTURES IN OFF-ROAD
VEHICLES

Cristian Bianchi

Advisor

Prof. Roberto Passerone
Università di Trento

Co-Advisor

Dott. Rosario Merlino
Iveco Group

September 2024

Abstract

In the context of automotive engineering, electrical and electronic (E/E) architectures are becoming increasingly crucial to vehicle development. The challenges posed by new vehicle functionalities, driven by current regulatory requirements and future applications tailored to autonomous driving capabilities, call into question existing design methodologies and highlight the need for a design paradigm shift to accommodate the high data rate demands of these new functionalities. From an industrial view-point, it is essential to design E/E architectures not only from a communication based perspective but also by considering other systems (electrical, mechanical, etc.) and non-technical requirements such as cost, risk, and time-to-market. In this dissertation, we present a tool for performing automatic design space exploration for E/E architectures in off-road vehicles, aimed at overcoming the limitations of current design processes and system architectures. Our approach not only addresses academic perspectives but also integrates the aforementioned industrial requirements. The effectiveness of our framework is validated through case studies of industrial relevance.

Contents

Abstract	i
1 Introduction	1
1.1 Background, Motivation and Objectives	1
1.2 Contributions	4
1.3 Outline	7
2 Related Work	9
2.1 Architectural Exploration Methodologies	9
3 Instruments and Background	19
3.1 In-Vehicle Network Communication Standards	19
3.1.1 Local Interconnected Network (LIN)	20
3.1.2 Controller Area Netowrk (CAN)	22
3.1.3 FlexRay	24
3.1.4 Media Oriented System Transport (MOST)	26
3.1.5 Low-Voltage Differential Signaling (LVDS)	27
3.1.6 Power Line Communication Interface (PLC)	27
3.1.7 Automotive Ethernet	28
3.2 Drive-by-Wire Technologies	33
3.3 E/E Architecture Topologies	34
3.3.1 Distributed E/E Architectures	35
3.3.2 Domain-Centralized E/E Architectures	36
3.3.3 Zonal-Centralized E/E Architectures	37
3.4 Simulation Tools	38
3.5 Cutting-Edge Technologies Selection	40
4 Methodology	43
4.1 Combining Optimization and Simulation	43
4.2 Optimization Methodology	45
4.2.1 Modeling of System Architecture and Mapping	45
4.2.2 Cost Function	53
4.2.3 Interconnection Constraints	53
4.2.4 Mapping Constraints	54
4.2.5 Rate, Cost, Weight, and Power Consumption Constraints	55

4.2.6	Linearization Techniques	56
4.2.7	Problem Formulation	57
4.3	Simulation Parameters and Metrics	58
4.3.1	CAN Bus Load	58
4.3.2	Camera Data Rate	58
4.3.3	Latency and Jitter	59
4.3.4	Data Payload	59
4.4	Constraints Reformulation	60
4.4.1	Adaptive Gradient Ascent Algorithm	61
4.4.2	Golden-Section Search	62
4.5	Trade-Off Evaluation	63
4.5.1	Non-Dominated Sorting Genetic Algorithm II (NSGA-II)	64
4.5.2	Multi-objective Bayesian optimization (MOBO)	66
5	Design Space Exploration Framework	69
5.1	Conceptual Structure of the Framework	69
5.2	Automated Design Toolbox Architecture	72
5.2.1	Software Components	73
5.3	Simulation Framework	79
5.3.1	TTAVBEthernetHost	80
5.3.2	TTAVBEthernetSwitch	85
5.3.3	BusEthernetGateway	87
5.3.4	Complete Network Architecture Simulation	89
5.4	Batch Interface	93
6	Evaluation and Results	95
6.1	DSE for Off-Road Vehicles and GSR Requirements	96
6.1.1	Problem Definition	96
6.1.2	Optimization Design Results	97
6.1.3	Design Results Validation	99
6.1.4	Discussion	102
6.2	DSE for Indoor Robotics Networks	104
6.2.1	Background	104
6.2.2	Problem Formulation	106
6.2.3	Automating the Simulation	108
6.2.4	Evaluation and Results	109
6.2.5	Discussion	117
6.3	Combining Optimization and Simulation for ORVs Architectural Design	118
6.3.1	Problem Formulation	119
6.3.2	Architectural Template and CAN Gateway Design	119
6.3.3	System Requirements and Evaluation Setup	123
6.3.4	System Bandwidth Optimization	124
6.3.5	System Power Consumption Optimization	129
6.3.6	Effects of AES Encryption	131

6.3.7	Effects of CAN Aggregation Strategy	135
6.3.8	Pareto Fronts	137
6.3.9	Scalability	141
6.4	Discussion	144
7	Conclusions	145
7.1	Summary of the Work	145
7.2	Impacts and Main Results	148
7.3	Discussion and Future Deployments	151
7.3.1	Reliability and task Mapping	151
7.3.2	Computational Complexity	151
7.3.3	Cyber Security Requirements	152
7.3.4	Transition Towards Fully centralized Architectures	158
	Acronyms	159
	Bibliography	167

List of Tables

2.1	Architectural Exploration Methods Comparison	16
3.1	SAE Classes for In-Vehicle Networks	19
3.2	Requirements for Semi-Autonomous Vehicles [46]	41
3.3	Comparison of Technologies for In-Vehicle Networks	41
4.1	Advantages and Disadvantages of Optimization and Simulation	45
6.1	Total System Cost, Power Consumption and Weight for Maximizing Data Rate (Scenario 1) and Minimizing Bandwidth Occupation at Backbone (Scenario 2).	99
6.2	Mean and Maximum Values for Latency and Jitter for Scenario 1 (Sim 1) and Scenario 2 (Sim 2).	101
6.3	CAN and Ethernet Network Load for Scenario 1 (Sim 1) and Scenario 2 (Sim 2).	101
6.4	Expected QoS RQMTs for URLLC applications	105
6.5	Robot Trajectories and Paths in First Scenario	111
6.6	Robot Trajectories and Paths in Second Scenario	111
6.7	Throughput and E2E Delay for Indoor Robotics Scenario	112
6.8	Robot Trajectories and Paths in Third Scenario	115
6.9	Gateway Forward Table: J1939 CAN messages (information and transmission rates) which need to be routed to the Ethernet domain to meet GSR requirements (SAS and TCO are the steering angle sensor and tachograph ECUs, respectively).	121
6.10	Extended Simulation Results of the Optimized E/E Architecture for Maximized BW Constraint	129
6.11	Extended Simulation Results of the Optimized E/E Architecture for Minimized Energy Consumption	131

List of Figures

3.1	LIN Bus Topology	21
3.2	LIN Bus Frame	21
3.3	CAN Bus Topology	22
3.4	CAN Bus Frame	23
3.5	FlexRay Line Topology	25
3.6	MOST Topology	26
3.7	PLC Topology	27
3.8	TT-Ethernet Switch Architecture and Frame Transmission	29
3.9	IEEE 802.1Qav Credit Base Shaping	31
3.10	IEEE 802.1Qav Credit Base Shaping Algorithm	32
3.11	SRP Mechanism	33
3.12	Distributed E/E Architecture	35
3.13	Domain-Centralized E/E Architecture	36
3.14	Zonal-Centralized E/E Architecture	37
3.15	SignalsAndGateways Framework [67]	40
4.1	Architectural Exploration Methodology	44
4.2	Example of System Architecture Selection	47
4.3	Example of Mapping	49
4.4	Example of Path	50
4.5	Example of Functional Link	52
4.6	Example of Valid and Invalid Functional Flows	52
4.7	Example of Correct and Incorrect Mapping Scenarios	55
5.1	Toolbox Conceptual Structure	70
5.2	Conceptual Software Architecture	72
5.3	Simulation Environment	79
5.4	TTAVBEthernet Host .ned Model (GUI)	80
5.5	TTAVBEthernet Switch .ned Model (GUI)	86
5.6	TTAVBEthernet Gateway .ned Model (GUI)	88
5.7	GUI Example for Full Network Configuration	89
5.8	BatchIF	93

6.1	Level of Development of the Framework in the Current Evaluation: The optimization process is fully automated, while simulation and constraint reformulation are fulfilled manually.	96
6.2	System Architecture and Mapping Optimization Results for Maximizing Data Rate (Left) and Minimizing Bandwidth Occupation at Backbone (Right).	98
6.3	Simulated Network Latency Results for Maximizing Data Rate (a) and Minimizing Bandwidth Occupation at Backbone (b).	100
6.4	Simulated Network Jitter Results for Maximizing Data Rate (a) and Minimizing Bandwidth Occupation at Backbone (b).	100
6.5	Level of Development of the Framework in the Current Evaluation: The optimization and simulation process is fully automated, while constraint reformulation is conceptually introduced	104
6.6	Framework Extension for Automated Simulation	109
6.7	Generated Architectures from ArchEx, for One Robot Scenario (a) and Two Robots Scenario (b).	110
6.8	Second Scenario from OMNeT++ Graphic User Interface (Qtenv) . . .	111
6.9	Throughput in Scenario 1 With One Robot	112
6.10	E2E Delay in Scenario 1 With One Robot	113
6.11	Throughput in Scenario 2 With Two Robots	113
6.12	E2E Delay in Scenario 2 With Two Robots	114
6.13	Third Scenario Floor Plan and Trajectories	114
6.14	Throughput in Scenario 3	115
6.15	E2E Delay in Scenario 3	116
6.16	Level of Development of the Framework in the Current Evaluation: The optimization, simulation and constraint reformulation process is fully automated	118
6.17	Architecture Conceptual Scheme.	120
6.18	Transmission Selection Algorithm for AVB and Time-Triggered Traffic: CBS uses clock and schedule, enabling TT communication with no interference	122
6.19	CoRE-RG Framework: Simulated System Architecture	123
6.20	Number of Simulations Required for Gradient-Ascent and Golden-Section Search Across Various Scenarios (Maximum Bandwidth Constraints Ranging from 50% to 90%)	125
6.21	Maximum bandwidth constraint update during iterations, with the gradient ascent with adaptive learning rate and the golden-section search (peak at 63%)	126
6.22	Maximum bandwidth constraint update during iterations, with the gradient ascent with adaptive learning rate and the golden-section search (peak at 85%)	126
6.23	Optimized E/E Architecture for Maximized BW Constraint (Colored Components are Those Selected by the Optimizer)	127

6.24	Optimized Mapping for Maximized BW Constraint (Only Sensor Library Shown)	127
6.25	Stream Latencies of the Optimized E/E Architecture for Maximized BW Constraint	128
6.26	Optimized E/E Architecture for Minimized Power Consumption (Colored Components are Those Selected by the Optimizer)	129
6.27	Optimized Mapping for Minimized Power Consumption (Only Sensor Library Shown)	130
6.28	Stream Latencies of the Optimized E/E Architecture for Minimized Power Consumption	130
6.29	Optimized E/E Architecture for Maximized BW Constraint (Colored Components are Those Selected by the Optimizer) when Applying AES Encryption	132
6.30	Optimized Mapping for Maximized BW Constraint (Only Sensor Library Shown) when Applying AES Encryption	132
6.31	Stream Latencies of the Optimized E/E Architecture for Maximized BW Constraint when Applying AES Encryption	133
6.32	Optimized E/E Architecture for Minimized Power Consumption (Colored Components are Those Selected by the Optimizer) when Applying AES Encryption	134
6.33	Optimized Mapping for Minimized Power Consumption (Only Sensor Library Shown) when Applying AES Encryption	134
6.34	Stream Latencies of the Optimized E/E Architecture for Minimized Power Consumption when Applying AES Encryption	135
6.35	Latency and Jitter Analysis for Different Hold-Up Factor Values	136
6.36	Pareto Fronts in the Case Study for Optimizing System Power Consumption	137
6.37	Pareto Analysis: 3D Plot of Power Consumption, Weight and Cost	138
6.38	Pareto Analysis: 3D Plot of Weight, Power Consumption and Cost	139
6.39	Comparison of 3D Pareto Surfaces obtained using the NSGA-II Algorithm, highlighting trade-offs between Power Consumption, Weight, and Cost.	140
6.40	Comparison of 3D Pareto Surfaces obtained using the MOBO Algorithm, highlighting trade-offs between Power Consumption, Weight, and Cost.	140
6.41	Scalability Analysis for Different Maximum Link Capacity Scenarios	142
6.42	Figure from [111]. Sensor Type, Positioning and Relation With Autonomous Driving Functions	143
6.43	Number of Sensors and Connection Capabilities vs. Total Resolution Time	144
7.1	TARA: Attack Scenario	154
7.2	Definition of Cybersecurity Properties	154
7.3	Subset of Damage Scenarios and Impact Rating	155
7.4	Threat Scenario Identification (Subset)	155
7.5	Attack Path Analysis and Feasibility Rating for <i>ID_THR_001</i>	156

List of Figures

7.6	Risk Value Determination	156
7.7	TARA: Attack Scenario with Countermeasure	156
7.8	Attack Path Analysis and Feasibility Rating for <i>ID_THR_001</i> with Countermeasure	157
7.9	Risk Value Determination with Countermeasure	157
7.10	Conceptual Smooth Transition Towards Fully Zonal-Centralized Architectures	158

Chapter 1

Introduction

This chapter introduces the problem we are addressing, beginning with a concise overview of the background in the off-road domain, highlighting the motivation behind our work and outlining our primary objectives. Following this, we summarize our contributions and provide an outline of the dissertation.

1.1 Background, Motivation and Objectives

Vehicle design is a process involving several technical aspects such as mechanics, hydraulics, electrical and electronics. In recent years, the significance of vehicle design has grown, particularly in the realm of electrical and electronic systems, to enhance safety and comfort, reduce emissions, and accommodate the increasing number of features demanded by the market. The new challenges that vehicle Original Equipment Manufacturers (OEMs) are facing in the design of Electrical and Electronic (E/E) architectures concern many aspects, ranging from the introduction of Advanced Driver Assistance Systems (ADAS) to the integration of systems for autonomous driving. Compared to the commercial car market, the off-road sector is a difficult niche where the integration of ADAS and the autonomous driving design is complex for several reasons.

The European General Safety Regulation (GSR) [66] states that, starting from 2024, an Off-Road Vehicle (ORV) must comply with several features in order to be sold on the market. These include, for instance, AIF (Alcohol Interlock Facilitation) [21], BSIS (Blind Spot Information System) [94], CSMS (Cyber Security Management System) [95], DDAW (Driver Drowsiness & Attention Warning) [23], ISA (Intelligent Speed Assist) [22], MOIS (Moving-Off Information Signal) [97], REV (Reverse Detection) [96], and TPMS (Tyre pressure monitoring Heavy duty) [93]. A first consequence is therefore

the need to integrate high data rate sensors, new Electronic Control Units (ECUs) and Drive-by-Wire (DBW) technologies [88, 4]. However, the existing E/E architectures for off-road vehicles, which mainly rely on distributed systems using Local Interconnect Network (LIN), Controller Area Network (CAN) and J1939 standards for heavy-duty vehicles [78]) communications, lack the necessary bandwidth to support the sensors required for implementing advanced features in both current and future markets, meaning that they have reached their load limit. At the same time, it is not industrially acceptable to redesign the entire vehicle infrastructure, due to excessive costs and high risk [58].

In addition to meeting regulatory Requirements (RQMTS), off-road vehicles must fulfill several high-level criteria, such as high load capacity and the ability to operate in harsh environmental conditions like steep slopes, rough terrain, and fords. These demands lead to the introduction of specialized systems not found in conventional cars, to ensure that the vehicle maintains high reliability and guarantee driving performance also in different environmental conditions. These systems include, for example:

- Transfer Case (TC): Transmits power from the transmission to both the front and rear axles to perform all-wheel drive functionality, besides enabling the implementation of various gear ratios, including high and low gears, to enhance navigation in off-road environments.
- Differential-Lock (DL): Prevents the movement of the front and/or rear differential causing wheels-slip, mainly to prevent the vehicle from becoming immobilized when the wheels lose traction.
- Power-Take-Off (PTO): To perform auxiliary functions without needing an additional engine, such as vehicle recovery via winch, air compressor system management, trailer handling capabilities, and secondary pump control.
- Central Tyre Inflation System (CTIS): To electronically adjust tire pressure based on the specific terrain, eliminating the need for manual tire inflation or deflation. Reducing the air pressure in a tire increases the contact area between the tire and the ground, enhancing traction and facilitating easier navigation on softer terrain.

Other conventional ECUs such as Body Computer (BC), Engine Control Module (ECM) and Electronic Transmission Control (ETC) are also integrated, but with specialized features for the off-road environment.

Another issue lies in the design process, which frequently relies on existing technological solutions that have been adopted and validated in other projects to minimize the risk of incorrect implementations. The initial phase involves defining system requirements, followed by development and validation using the V-model methodology [35]. Generally speaking, many tests are conducted on a physical prototype, allowing for only a few iterative adjustments in the E/E architecture, thereby limiting the exploration of alternative architectural solutions. While initially effective, this approach will become less viable as future vehicle deployments demand increasingly advanced functionalities, leading to worse time to market, performance, cost and reliability [103].

The off-road domain has limited research due to its niche nature. Consequently, much of the existing literature focuses on the traditional automotive sector. For example, numerous studies focus on autonomous driving systems and potential architectural solutions [9, 59, 76, 62, 56]. However, finding examples and studies on the integration and validation of these systems within a real vehicle is quite challenging. Electrical and electronic research in the off-road sector (including robotic platforms) primarily addresses specific application challenges, such as road recognition and path planning [30, 100, 91, 84, 109, 38], direction prediction [81], obstacle detection [104, 33, 11], vehicle recognition [39], sensors [110, 105, 87], and communication techniques [44]. Despite this, these studies focus on specific functionalities and neglect the broader vehicular architecture and system-level design concerns, such as topology and system implementation, which are essential for developing real off-road vehicles and conducting analysis related to real industry case studies.

Considering the lack of specific examples in the literature on E/E design for the off-road domain, it is advisable to reference design paradigms from other Cyber-Physical System (CPS) fields, such as traditional automotive systems, aircraft, industrial robotics, and industrial automation. Within this context, the adoption of analytical models and simulation tools becomes crucial in the design process. These tools facilitate the evaluation of various architectural configurations and enable the early identification and exclusion of unfeasible solutions, thereby optimizing the design workflow and ensuring robust system integration. Therefore, a paradigm shift towards Design Space Exploration (DSE) is advisable for current and future projects.

Given these premises, one of the objectives of this dissertation is to identify and adopt the most appropriate communication protocols and architectural solutions for industry-driven constraints and considerations. This involves a deep analysis of existing protocols and architectures to determine their suitability for enhancing the accuracy and reliability of project outcomes. Additionally, the dissertation aims to explore alternative

and complementary design methods that can further improve these outcomes, thereby increasing market competitiveness.

In addition, we aim to develop and test an automated design approach based on DSE. This approach is intended to significantly reduce the cost and time associated with correcting design errors that typically arise after the construction of physical prototypes. By automating the design process, the dissertation seeks to minimize human error.

Finally, we aim to validate the proposed methodology through case studies of industrial relevance. This involves a detailed analysis of the results based on critical system design parameters.

To summarize, our key objective is the development of a framework for design space exploration of E/E architectures in off-road vehicles which reaches the above mentioned goals, thus providing a complete tool for the design and validation of E/E architectures given input specifications which include different system requirements.

The dissertation will also outline a strategic road-map for transitioning from current architectural frameworks to more advanced future architectures, ensuring a seamless integration and adoption of new technologies.

1.2 Contributions

Our work consists of the development and test of a novel design methodology and framework for next-generation E/E off-road vehicles architectures.

By leveraging cutting-edge technologies and communication protocols, we adapted and extended an existing framework [41] for Design Space Exploration (DSE) to our specific application domain, namely off-road E/E vehicle architectural design. We enhanced this framework by integrating optimization and simulation techniques through an in-the-loop iterative methodology. We conducted a deep analysis of regulatory requirements and derived system-level design parameters to facilitate integration testing via a simulator. Initially, we evaluated the optimization results through automated simulation and data retrieval across two industrially significant case studies: one involving off-road vehicles and the other focusing on indoor robotics networks. Subsequently, we extended this evaluation using an in-the-loop methodology for a larger case study again in the off-road domain. This included comprehensive system-level analysis (e.g., end-to-end latency, jitter, bandwidth occupation, effects of encryption), the generation of Pareto fronts to analyze trade-offs between system cost, weight and power consumption, and a qualitative scalability analysis.

Tools Selection and Evaluation. After an in-depth study of the state-of-the-art, we assessed various communication standards and vehicular architectures, selecting the most suitable ones for the high data rate channels required by new vehicle functionalities. In particular, Audio Video Bridging (AVB) and Time-Triggered Ethernet (TT Ethernet) were selected due to their protocol characteristics, which are well-suited to addressing our specific requirements. We identified Mixed-Integer Linear Programming (MILP) as the most suitable optimization technique for solving our system architecture and implementation challenges. We determined that the CoRE-RG¹ framework was the most appropriate for our task, as it supports mixed-criticality network simulation (CAN and Ethernet) and allows integration with the optimization tool framework.

Requirements Analysis. From the GSR requirements analysis, we derived a comprehensive requirement matrix to facilitate the integration of new functionalities into the vehicular architecture. This step is critical for system simulation, as it identifies the specific CAN messages that transport essential vehicular data (e.g., vehicle speed, gear ratio, engine speed, steering angle position). These messages must be read by the logic governing the new functionalities. This detailed mapping is crucial to assess the feasibility of system integration, ensuring that the new functionalities can be seamlessly incorporated without affecting existing operations. Regarding industrial requirements, as outlined in Section 1.1, redesigning the entire architecture is not feasible due to economic and time constraints. Consequently, we have adopted an architectural design strategy that preserves the existing architecture while integrating new paradigms for the high data rate system. During optimization, our objective is to determine the optimal topology for the high data rate domain and validate the integration of the entire architecture through simulation. The CAN messages can be routed to the Ethernet domain via dedicated gateways. In this context, we provide a design strategy for the gateway, detailing which messages need to be routed and at what rate, in accordance with GSR requirements.

Framework Extensions. As anticipated, we have developed a CPS exploration optimizer based on the ArchEx tool [63, 43], extending and adapting it for the requirements of the project. ArchEx is a MATLAB toolbox that uses YALMIP² as the interface to the solver. However, YALMIP is no longer supported by recent versions of MATLAB and therefore we rewrote the framework in Python, taking the basis constraints needed for our project and adding the required new ones. Instead of YALMIP,

¹<https://github.com/CoRE-RG/CoRE4INET>

²<https://yalmip.github.io/>

we decided to use the CVXPY³ Python library to perform MILP modeling and solving (via the CPLEX or SCIPY optimizer). We adapted and extended the framework to deal with a zone-based E/E topology for off-road-vehicles, a drive-by-wire guidance system, and a heterogeneous network including CAN, AVB, and TTEthernet protocols. The structure of the optimization tool can be summarized as a set of templates that are used to model design requirements.

Combination of Optimization and Simulation. Our core contribution consists in the integration of *optimization* and *simulation* techniques. This is beneficial, because during the design process there are real-world deployment aspects that are too complex and hard to model for an optimizer, or that simply lead to excessive run-times. Conversely, simulation is very effective at handling lower-level details such as protocol handshakes and run-time collisions. Moreover, it allows us to test the integration of the optimized domain into the entire vehicular architecture, verifying the coexistence of different standards such as CAN and Ethernet. On the other hand, simulation does not scale to a large design space exploration, and only a sub-set of architectures can be effectively evaluated. The idea we pursue in this work is therefore to combine optimization and simulation. The optimizer selects candidate architectures from a large design space, using relatively abstract models, excluding unfeasible solutions. The simulator therefore examines only a restricted subset, extracting detailed and accurate performance parameters to validate the selection. These parameters are then fed-back to the optimizer to refine the solution. The procedure is iteratively repeated to converge to a final suitable architecture, thereby maximizing the benefits derived from both techniques. In other words, the optimizer is used to focus the attention to only viable solutions, pruning the design space, while the simulator evaluates the performance for accurate results. As a result, our work greatly improves design automation, where the optimizer generates candidate solutions through different patterns, and simulation results are automatically retrieved.

Improvements in Development Process and Industrial Implications.

From an industrial perspective, the development of a design space exploration framework allows us to provide tailored design solutions that enhance the current design phase.

Traditional processes often rely on extending existing solutions and past knowledge. However, our proposed methodology is designed to efficiently explore several design alternatives. Traditional methodologies typically lack also the exploration of different

³<https://www.cvxpy.org/>

design solutions due to cost and time constraints. Additionally, they face challenges in managing the design complexity introduced by current and future market requirements. A design exploration tool can address these issues by efficiently exploring the solution space, automating analysis and design (thus reducing human error), and arriving at feasible solutions more quickly. This results in a reduced time-to-market and an overall improvement in product quality.

The designers, in fact, can focus on high-level design requirements and use the tool to quickly generate feasible solutions, adjusting parameters to explore different design alternatives.

Using such a tool also reduces the risk of architectural changes during the design process, since even minor changes typically require extensive manual analysis and considerations, thus making the process inflexible. An automatic design space exploration approach can manage this issue, as project requirements can be easily adjusted during the design phase, and alternative solutions can be quickly generated with minimal effort. This leads to a reduction in risk and further improves time-to-market.

Finally, the use of both optimization and simulation can reduce the costs associated with constructing physical prototypes. Although building a physical prototype is a mandatory step, many integration problems can be identified in advance through simulation, since it allows to precisely estimate the physical behaviors of the communication and to test the entire vehicle E/E communication system.

1.3 Outline

The dissertation is structured as follows:

- **Related Work:** We begin with an overview of the state-of-the-art in domain space exploration techniques tailored to the automotive domain and other cyber-physical domains, contextualizing them with our design requirements. In this section, we highlight gaps in the literature and state the needs to achieve our design goals.
- **Instruments and Background:** This chapter summarizes the cutting-edge technologies that enable us to achieve our design objectives, including vehicular architectural possibilities, in-vehicle network technologies, and simulators.
- **Methodology:** In this chapter, we formally describe the methodology adopted. In particular, we explain the advantages of using a combination of both optimiza-

tion and simulation. We also detail the methodology adopted, provide a formal description of the performance metrics we use for validation, and describe the approach we use to reformulate the constraints for the optimizer.

- **Design Space Exploration Framework:** This chapter details the tool we developed, the automation of the simulation process, and the management of automatic interactions between the optimizer and the simulator to facilitate iterative simulations in the loop. We provide a deep explanation of the software architecture of the tool as well as the library modules used in the simulation environment.
- **Evaluation:** In this chapter, we present our main results. We report them following the three works we published, which represent the temporal improvements made to the toolbox. In the first work, we present an initial use of the framework to design and validate an architecture for the design of E/E architectures in off-road vehicles. In this case, the optimization was automated while validation and data retrieval were done manually. In the second work, we improve the design automation by also automating the simulation and data analysis, and we validated the tool over a different case study, i.e., industrial robotics network. We also proposed in this work the combination of optimization and simulation in an iterative manner, but only as a concept and not implemented. In the last work, we finalized the toolbox by fully automating the design process, integrating the combination of optimization, simulation, and iterative constraints reformulation. We report results of different case studies with different design objectives, including considerations about the introduction of cryptography in the design, the derivation of Pareto fronts, and a qualitative scalability analysis.
- **Conclusions:** Finally, we present our conclusions, key takeaways, and future research directions, explaining how the tool can be used for a gradual transition towards fully centralized architectures.

Chapter 2

Related Work

This chapter presents significant works related to the problem we are addressing. It provides a concise summary and discusses the methods that can be utilized in our research. Additionally, it identifies gaps in the literature, highlighting the need to develop or adapt existing frameworks to achieve our design goals.

2.1 Architectural Exploration Methodologies

The identification of both Design Space Exploration (DSE) capability and optimized topologies for Cyber-Physical System (CPS) is a crucial point for the future E/E automotive systems development to improve performance, reliability, power consumption, price, and time to market, since unfeasible architectural solutions can be eliminated in the early stages of design [103].

In general, DSE allows the designer to separate the required functionalities from the architecture and to model them through a mathematical model. To find an optimal (or sub-optimal) architectural solution, optimization techniques are applied considering both feasibility and system requirements. In the literature, there are numerous approaches to DSE for CPS ranging from the aerospace to the traditional automotive sector.

Joshi et al. [37] developed an Integer Linear Programming (ILP) methodology to solve the problem of task schedulability and end-to-end latency requirements satisfiability in an automotive Ethernet network. The design of the Time-Triggered (TT) Ethernet architecture involved a heuristic method divided into four main steps: clustering, mapping, scheduling, and latency computation. Clustering involves grouping tasks based on certain criteria to optimize overall system performance. Mapping assigns

these clustered tasks to specific network end systems, ensuring an efficient distribution of workload. Scheduling determines the timing of message transmissions to ensure all tasks meet their deadlines, adhering to the time-triggered nature of the system. Latency computation calculates the delay for each message to verify that the system meets the required performance metrics. This method was implemented using C++ for the core logic and CPLEX for solving the optimization problems involved in mapping and scheduling. However, this approach did not consider the complexities of multi-domain systems, where different subsystems might need to interact with each other. Additionally, it did not address the management of different system configuration variants, which is crucial for systems that need to adapt to varying operational conditions or requirements.

A multi-variant-based DSE is proposed by Graf et al. [27]. The method leverages Integer ILP for the concurrent synthesis of multiple Electronic Control Unit (ECU) variants, encompassing both hardware and software, tailored for various car configurations and models. The architectural exploration employs the Y-Chart approach, which distinctly separates functionality from architecture. This separation facilitates system synthesis concerning resource allocation, task binding, and message routing, effectively modeling multi-cast communication between ECUs within the system architecture. The primary objectives of the authors are to minimize both the monetary cost and the number of hardware variants required in the vehicular architecture.

An architecture modeling and exploration framework for autonomous and semi-autonomous driving was proposed by Zheng et al. [114]. The tool facilitates architecture evaluation and design by mapping software applications onto heterogeneous hardware platforms and various network domains. It analyzes and optimizes several design parameters, including timing, reliability, security, and performance. Software is modeled using a synchronous reactive task graph, with key timing analysis parameters being message length, message period, source, and destination. The hardware model is described using an architecture modeling and description language, focusing on CPU speed, number of cores, and scheduling policies. The tool evaluates task timing, message timing, and end-to-end path latency to ensure optimal performance.

McLean et al. address the challenge of mapping safety-critical Advanced Driver Assistance Systems (ADAS) applications onto modern multi-processor platforms. [55]. Their work primarily focuses on configuring hardware platforms to manage multiple inter-dependencies and stringent timing requirements of real-time tasks, ensuring reliable and efficient operations. They utilize the IEEE 802.1 Time-Sensitive Networking (TSN) deterministic Ethernet standard for communication. For scheduling optimiza-

tion, they propose two heuristic-based algorithms: simulated annealing and genetic algorithm. Given an ADAS platform and a list of tasks as inputs, the problem outputs include the mapping of unassigned tasks to cores and an optimal static schedule. This schedule ensures that timing constraints, such as offsets, deadlines, and jitter, are satisfied. The approach effectively balances the computational load while adhering to the stringent requirements of safety-critical ADAS applications.

Maier et al. propose a methodology for designing the power supply and wiring harness in an E/E vehicular architecture [54]. Given the vehicle geometry and a set of electric components, they use k-means clustering to assign electric components to specific vehicle zones. Additionally, they apply the Dijkstra algorithm to optimize cable routing within the vehicle. The researchers validate their results by integrating a simulation, demonstrating how it is possible to reduce the wiring harness through a correct allocation of zones. Additionally, they highlight functional limitations that may arise if the design is not executed correctly. This approach aims to enhance the efficiency and reliability of the electrical system in vehicles.

Askaripoor et al. propose a framework for multi-core embedded systems and communication networks. This framework automates the mapping of software components (applications, processes, and threads) to hardware components (processors, cores, and ECUs) in real-time scenarios [5]. They achieve this by automatically transforming problem requirements into constraints using Mixed-Integer Programming (MIP) and a model-driven development approach. The Gurobi solver serves as the optimization engine. The framework undergoes both design-time evaluation, considering mapping and scalability, and run-time evaluation on real hardware. The High-Performance Computing Unit (HPCU) is modeled using hardware such as the Nvidia Drive AGX Xavier Developer kit, an Intel i210-based evaluation board, and an STM32L476VG microcontroller.

Voss et al. describe the framework of Autofocus3, a model-based environment designed to assign software components to hardware components through DSE for multi-criteria problems in automotive E/E systems [101]. The tool provides a graphical user interface and allows users to find design solutions in a semi-automatic way using an Satisfiability Modulo Theory (SMT) solver. The proposed DSE approach considers various problem constraints, including deployment constraints related to the assignment of components to computation units, timing constraints that take into account concurrent execution and communication over a shared channel, and safety-related constraints to ensure the system meets necessary safety standards.

Höttger et al. have developed App4MC, a sophisticated tool designed for multi-core

and many-core systems [31]. This tool employs both hardware and software models to accurately define various system properties, including processor types, inter-module connection types, Operating System (OS) schedulers, and task-specific attributes such as execution times and deadlines. The platform, which is freely available under the Eclipse Public License, has proven to be highly effective in the automotive industry. It supports integration with a variety of external tools, enabling comprehensive system verification and simulation, particularly in the areas of scheduling and timing analysis. This adaptability ensures that the platform can be tailored to meet specific requirements, enhancing its utility in real-world applications.

Aleti et al. presented an Eclipse-based tool, named ArcheOpterix, designed to implement evaluation techniques and optimization heuristics for AADL (Architecture Modeling and Description Language) specifications, aimed at identifying suitable embedded system architectures based on quality and functional requirements [3]. This tool, a plugin for OSATE (Open Source AADL Tool Environment), includes an AADL model parser, an architecture analysis module, an architecture optimization interface, and an AADL model generator. The optimization process leverages an evolutionary algorithm, considering constraints such as data transmission reliability and communication overhead. The tool supports modeling various ECUs (Electronic Control Units), buses, and services. However, it does not address the multi-core mapping problem for computing platforms in automotive applications, nor does it cover aspects like reliability.

Pop et al. address the design challenges of distributed Cyber-Physical System (CPS) utilizing Time-Sensitive Networking (TSN) as the communication backbone, focusing on the scheduling of Time-Triggered (TT) flows and the routing of Audio Video Bridging (AVB) flows within a switched multi-hop network architecture [72]. The problem definition includes inputs such as the network topology and defined sets of AVB and TT flows. Constraints involve frame size and timing, bandwidth and latency limitations on network links, end-to-end deadlines and isolation requirements for flows, specific timing deadlines for frame delivery, and ensuring that TT and AVB frames do not interfere with each other. The outputs are the determined paths for AVB and TT flows through the network, allocation of TT frames to egress port queues, and schedules for gate operations to manage frame transmission. The proposed solution employs an Integer Linear Programming (ILP) formulation to optimize the scheduling and routing tasks, compared against existing methods that utilize Satisfiability Modulo Theory (SMT) formulations, highlighting the advantages and performance trade-offs of the ILP-based method.

A further approach for CPS architectural exploration was introduced by Kirov et al. [41] with the ArchEx toolbox. This framework models the architecture through a graph and uses Mixed-Integer Linear Programming (MILP) to solve the mapping problem, optimizing the system including timing and reliability constraints. The tool offers a comprehensive system configuration and mapping between virtual components and their real-world counterparts, enabling a real physical implementation of the system. It has been rigorously evaluated through two case studies: one focusing on an aircraft power distribution network and the other on a re-configurable production line. The framework was further extended to incorporate the failure probabilities of system components during the design phase, thereby enhancing the overall system reliability [63].

A proposed framework extension in the wireless sensor network domain aims to optimize topologies and component sizing. The goal is to jointly select network components, placement, and routes while minimizing an objective function under correctness guarantees. Specifically, the framework considers energy consumption, link quality, and localization requirements, which are expressed as mixed integer linear constraints over the network path. [42, 43]. The toolbox extension underwent testing on data collection and localization networks, demonstrating how the proposed approximate encoding method can handle larger problems effectively.

A DSE for a human intranet network was proposed by Moin et al. [60]. In their research, the authors adopt an iterative approach based on a Mixed-Integer Linear Programming (MILP) routine. This routine is interfaced with CPLEX via the PuLP Python interface. Additionally, they incorporate a discrete-event simulator using OM-NeT++. The primary objective is to identify candidate solutions that satisfy network lifetime and packet delivery ratio requirements.

Finn et al. [24] employ an iterative optimization-based approach for the design of an aircraft environmental control system. In their study, the researchers establish an iterative process that combines a discrete architecture selection engine (which uses CPLEX solver) with a continuous sizing engine. The discrete optimization routine proposes candidate architectures to the sizing engine. Subsequently, the sizing routine optimizes the continuous parameters using simulation to evaluate physical models and monitor requirements. To formalize design requirements, the team employs contracts expressed in terms of temporal logic constructs and mixed integer-linear constraints. These contracts serve as abstractions for component behaviors, allowing them to formalize system-level requirements and verify the correctness of compositions and refinements between components. Notably, their experiments demonstrate a significant reduction in design time by an order of magnitude when considering an industrial case study.

Nuzzo et al. propose a rigorous platform-based methodology for designing the topology of an aircraft electric power system [64]. The design process involves three key steps: topology synthesis, control synthesis, and simulation-based design space exploration and verification. To achieve their goals, they formulate a Mixed-Integer Linear Programming (MILP) that minimizes overall cost while satisfying connectivity, power flow, and reliability requirements. These requirements are expressed through linear arithmetic constraints on boolean variables and probabilistic constraints. The compositional approach relies on contracts to ensure independent implementation of system topology and control. The topology synthesis algorithm is implemented in Matlab and utilizes CPLEX to solve the MILP at each iteration.

In the context of embedded systems, Kugele et al. address the challenge of deploying software applications on appropriate execution units and defining connections among sensors and actuators [48]. They introduce a domain-specific constraints and optimization language to tackle industry-specific problems. Three solving strategies are presented: evolutionary algorithms (extension to the NSGA-II), SMT and ILP. Additionally, they define an approach that combines the first two methods. The objectives and constraints are defined using the System Architecture Optimization Language (SAOL), which simplifies modeling for engineers. The feasibility of their approach is evaluated through two case studies: a flight control system and a seawater desalination plant. They successfully find deployment solutions that satisfy all constraints with less required hardware, making the solutions more cost-effective. It is observed that SMT and ILP methods are more suitable for industrial-sized models.

Li et al. introduced a framework called Automated Quality-driven Optimization of Software Architecture (AQOSA) [50]. This framework integrates modeling technologies, performance analysis techniques, and advanced evolutionary multi-objective optimization algorithms to automatically enhance the non-functional properties of system architectures (NSGA-II, SPEA2, and SMS-EMOA). They validated the tool using two case studies: a car radio navigation system and a business reporting system. The goal was to determine a set of optimal architectures with respect to quality attributes such as processor utilization, cost, and data flow latency. The results demonstrated that the tool helps software architects reduce the workload for modeling and design. They also showed that while random search is sufficient for small design spaces, evolutionary-based algorithms significantly outperform random search in larger design spaces.

Koziolek et al. have introduced a framework, named PerOpteryx, that integrates architectural performance tactics into a metaheuristic optimization process using the multi-objective evolutionary algorithm NSGA-II [45]. This approach allows software ar-

chitects to focus on automatically determined optimal trade-offs between various quality attributes, enabling them to select the best options for their specific situations. The tactics used for improving software architecture quality encode design knowledge and best practices, including hardware, software, and network features. The tool was evaluated through two case studies: a distributed business reporting system and an industrial control system. The results demonstrated that the tool could identify meaningful Pareto fronts, effectively moving beyond the traditional trial-and-error approach to software architecture design. This method provides a more principled, engineering-based approach to architecture design by leveraging these tactics.

Given the various previously discussed approaches (summarized in Table 2.1), we have chosen the ArcheEx framework as the baseline for our project. This decision is based on its suitability for addressing our exploration problem. ArcheEx allows us to model the vehicle topology using a graph representation, which is crucial for capturing the complex interconnections and dependencies within the system. Additionally, ArcheEx provides a mechanism to define a library of components, enabling automatic selection of system implementations. By modeling components with various parameters of interest (such as cost, power consumption, weight, and data rate), we can perform efficient exploration and optimization. We express system requirements as linear constraints and rules within the ArcheEx framework, ensuring that our design meets specified criteria. Moreover, ArcheEx supports exploration of redundant topologies, allowing us to identify an optimal topology aligned with our design goals. In summary, ArcheEx provides a comprehensive toolkit for tackling our exploration problem, combining graph-based modeling, component libraries, parameterization, and efficient optimization. Leveraging these features, we aim to achieve our design objectives effectively.

Observing Table 2.1, it is evident that fewer studies combine simulation with optimization to verify results. In our case, the verification step is fundamental to assess the feasibility of the optimization and to correct the design before building the physical prototype, thereby reducing the overall design costs.

Regarding techniques for automotive E/E architectural optimization, most works focus on mapping software components to hardware or processing units. While this task is important, it often overlooks the workload of connections and the selection of sensors/actuators within the topology, given a set of Commercial-Off-The-Shelf (COTS) components available in the market. Factors such as cost, weight, and power consumption are not adequately addressed.

Moreover, these studies do not consider the integration of the solution within the

2.1. Architectural Exploration Methodologies

Table 2.1: Architectural Exploration Methods Comparison

Reference	Application Domain	Design Objectives	Optimization Techniques	Verification Method
Joshi et al. [37]	Automotive E/E architecture	Automotive Ethernet task schedulability and end-to-end latency requirements satisfiability	ILP + CPLEX	Analytical
Graf et al. [27]	Automotive E/E architecture	Minimize both the monetary cost and the number of HW variants	ILP	Analytical
Zheng et al. [114]	Automotive E/E architecture	Mapping SW to HW given timing, reliability, security, and performance constraints	ND	Analytical
McLean et al. [55]	Automotive E/E architecture	Mapping safety application onto multi-processor platforms and optimal schedule given timing constraints	SA & GA	Analytical
Maier et al. [54]	Automotive E/E architecture	Power supply and wiring harness design to optimize cable routing and electric component selection	K-means clustering + Dijkstra algorithm	Analytical + Simulation
Askariipoor et al. [5]	Automotive E/E architecture	Mapping SW to HW given safety and timing constraints	MIP + Gurobi	Analytical + Prototype
Voss et al. [101]	Automotive E/E architecture	Mapping SW to HW given timing, communication and safety constraints	Meta Search + SMT	Analytical
Höttger et al. [31]	Automotive E/E architecture	HE & SW Codesign	ILP + GA	Analytical + Simulation
Aleti et al. [3]	Embedded systems architecture	Mapping SW to HW given timing, network and reliability constraints	Evolutionary algorithms	Analytical
Pop et al. [72]	Automotive E/E architecture	Scheduling and routing of automotive Ethernet flows given timing and network constraints	ILP + CPLEX	Analytical
Kirov, Nuzzo et al. [41, 63, 42, 43]	Aircraft power network, re-configurable production line, wireless sensor networks	Architectural Selection and component mapping given timing, network and reliability constraints	MILP + CPLEX	Analytical
Moin et al. [60]	Human intranet networks	Architecture selection given network constraints	Iterative optimization (MILP + CPLEX) + simulation (OmNET++)	Analytical + Simulation
Finn et al. [24], Nuzzo et al. [64]	Aircraft control system	Electric power systems topology synthesis	Iterative optimization (MILP + CPLEX) + simulation	Analytical + Simulation
Kugele et al. [48]	Flight control system, seawater desalination plant	Mapping SW to HW and topology selection	Evolutionary algorithms, ILP, SMT	Analytical
Li et al. [50]	Car radio navigation system, business reporting system	Optimal architectures selection given processor utilization, cost, and latency requirements	Evolutionary algorithms	Analytical
Koziolek et al. [45]	Business reporting system, industrial control system	Improving software architecture quality	Evolutionary algorithms	Analytical

entire E/E system. Verification of the complete system design is crucial to assess its feasibility. For instance, we have a design constraint that an architecture satisfying network constraints derived from autonomous driving and ADAS functionalities must not necessitate a complete redesign of the vehicle architecture. This means the current topology, consisting of a CAN bus, must remain unchanged and be considered a fixed part of the design.

In this dissertation, we aim to extend the toolbox to optimize a topology that supports new functionalities and to verify them not as standalone components, but as integrated parts of the existing architecture. To achieve this, we need to extend the

framework to work with new constraints, rules, and modules for optimization related to our design needs. Additionally, we require a simulation tool that considers the existing CAN architecture and models the interaction between the existing and new domains.

Finally, another innovative aspect lies in how we combine optimization and simulation in our methodology. An optimizer can quickly converge to a candidate solution. However, some aspects of real deployment, such as protocol handshakes, real-time scheduling, and collisions, are very difficult to model. Representing these as linear constraints and rules could be either impossible or can lead to excessive run-times or impractical optimization times due to the generation of a vast number of parameters.

On the other hand, simulation can precisely estimate design parameters like network load and delays, and accurately model the behavior of the network, including buffers, network ports, and cable connections. However, relying solely on simulation requires extensive trial and error by the designer, which can be time-consuming. Additionally, it may be challenging to scale to a large design space, potentially leaving some parts of the design space unexplored.

Therefore, we aim to combine optimization and simulation to leverage the benefits of both methods. We model only certain system behaviors through the optimizer to find candidate solutions. Then, we use the simulator to extract more precise results to assess system performance and check if the resulting implementation is feasible within the final architecture. If the solution is not feasible, we refine the constraints and rerun the optimization iteratively until convergence is reached and our design goals are satisfied.

Chapter 3

Instruments and Background

In this chapter, we review the current technologies and concepts related to the development of Electrical and Electronic (E/E) vehicle architectures. This includes communication standards for in-vehicle networks, Drive-by-Wire (DBW) technologies, topology solutions, and simulation tools. After selecting the design method, this section is crucial as it lays the foundation for constructing a practical template that can be modeled through the design tool and which can be implemented into a real vehicle infrastructure, to support autonomous driving and Advanced Driver Assistance Systems (ADAS) features.

3.1 In-Vehicle Network Communication Standards

Society of Automotive Engineers (SAE) classified in-vehicle networks in four main classes (A, B, C and D class) which are based on the speed or bandwidth of network, safety critical relevance and real time response delay [36]. An overview of the data rates, response times, and application domains of the various classes is provided in Table 3.1.

Table 3.1: SAE Classes for In-Vehicle Networks

Feature	Class A	Class B	Class C	Class D
Data Rate	up to 10 kb/s	10 kb/s to 125 kb/s	125 kb/s to 1 Mb/s	>>1 Mb/s
Response	150 ms	Very Fast	Very Fast	Very Fast
Application	Door open/close/lock, weather guideline, windows glass, parking light, indicator light, vehicle Body	Auto gear system, ignition control of engine, braking system.	Brake system through wire, steer system through wire, gear shift through wire	Audio/video systems which are used for entertainment, satellite navigation system, mobile calling/ receiving, safety systems

- **Class A** is a low speed network characterized by event-driven message transmis-

sion. This type of network supports up to 10 kb/s and the transmission delay falls within 50 ms to 150 ms. Typical applications are related to body domain like light controlling, door open/close, door window open/close, seat positioning, window-washers control, light sensor acquisition.

- **Class B** is a medium speed network which supports event driven, periodic message transmission with sleep/weak up calls. The data rate is from 50 kb/s to 125 kb/s and does not support safety-critical applications. The main use of this network is to interconnect different systems in the vehicle.
- **Class C** is a high communication network (data rate from 125 kb/s to 1 Mb/s) with very low latency. It can be utilized for a subset of drive-by-wire applications that require medium performance, as well as for vehicular communication between various main ECUs, such as the Electronic Transmission Control (ETC), Body Computer (BC), Instrument Cluster (IC), Engine Control Module (ECM), and others.
- **Class D** supports rates over 1 Mb/s and has a strong latency constraint. It supports transmission of multimedia data and ADAS applications (which require bandwidth from 10 Mb/s to 30 Mb/s). Other typical applications are Drive-by-Wire (DBW) systems (Steer-by-Wire (SBW), Brake-by-Wire (BBW) etc.) which require higher performance.

In the following sections, we will examine the key features of various communication standards used in in-vehicle networks. This analysis will cover aspects such as data rates, network topologies, and the advantages and disadvantages of each standard. In the final section, we will discuss the process of selecting the most appropriate communication standards for our industrial application, focusing on those that enable high data rate links.

3.1.1 Local Interconnected Network (LIN)

Developed in the late 1990s by five automakers (BMW, Volkswagen Group, Audi, Volvo Cars, and Mercedes-Benz) and supported by hardware and networking technologies from Volcano Automotive Group and Motorola, the Local Interconnect Network (LIN) is a serial master-slave protocol, with its first version released in November 2002. It is characterized by a single-wire connection, low cost, and reduced energy consumption (it supports sleep mode). The protocol supports up to 16 slaves (practically up to

7) without arbitration ID and utilizes parity bits and checksums to detect errors in transmitted frames. It operates at a nominal voltage of 12 V and uses binary symbols for communication: a dominant '0' and a recessive '1'. Slave Electronic Control Units (ECUs) can only transmit data after being queried by the Master ECU, which sends a corresponding header. The communication schedule is predefined during the design phase and is organized in a schedule table. This table contains a list of frames to be transmitted and their respective time slots, known as Frame Slots, within which each frame is sent. A typical topology for LIN is shown in Figure 3.1.

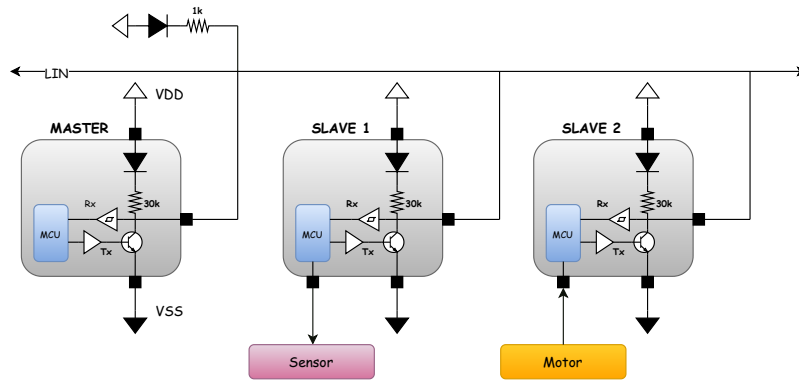


Figure 3.1: LIN Bus Topology

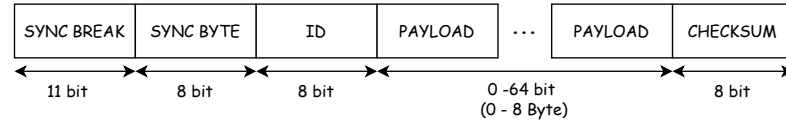


Figure 3.2: LIN Bus Frame

Each LIN frame contains the following fields (Fig 3.2):

- **Synchronization break:** This field notifies all nodes that a message is being transmitted. It is always sent by the master node.
- **Synchronization byte:** The synchronization field is always transmitted by the master node to ensure all nodes are synchronized.
- **Identifier byte:** The message identification field, which is uniquely defined for up to 64 IDs, is always transmitted by the master node.
- **Payload:** The data field can be transmitted by either the master node or the slave nodes.

- **Checksum byte:** The error detection field consists of 8 bits and can be transmitted by either the master node or the slave nodes.

Since it is a low-speed protocol (up to 20 kb/s), it can be used in Class A applications of the SAE standard [36]. The LIN frame can carry 1/2/4/8 Bytes of data and the bus length is up to 40 m.

3.1.2 Controller Area Network (CAN)

Introduced in 1983 at Robert Bosch GmbH and first released in 1986 at the Society of Automotive Engineers (SAE) conference in Detroit (Michigan), the Controller Area Network (CAN) is a multi-master vehicle bus standard utilizing two shielded cables to facilitate communication among microcontrollers, ECUs, and other devices via message-based protocols. There are two primary variants of CAN: low-speed CAN, which operates at data rates ranging from 10 kb/s to 125 kb/s, and high-speed CAN, which supports data rates from 125 kb/s to 1 Mb/s. Typically, low-speed CAN is employed in Class B SAE applications, while high-speed CAN is utilized for Class C SAE applications, which are associated with safety-critical functions. Each CAN frame consists of 8 Bytes, and the error-checking mechanism is 16 bits in length. The CAN protocol employs the Carrier Sense Multiple Access - Collision Detection (CSMA-CD) access control method, which is both priority-driven and decentralized. The maximum bus length can extend up to 40 m, contingent upon the application rate. While the implementation cost of CAN is higher than that of a Local Interconnect Network (LIN), it remains lower than that of a FlexRay network. However, it is important to note that CAN is not ideally suited for safety-critical real-time systems. A typical CAN topology is shown in Figure 3.3.

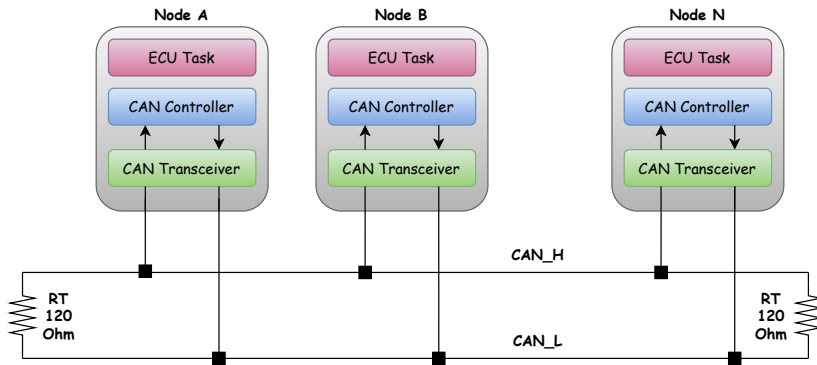


Figure 3.3: CAN Bus Topology

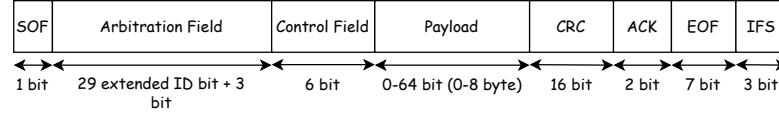


Figure 3.4: CAN Bus Frame

It is possible to see that the CAN network must be terminated with two $120\ \Omega$ resistors to minimize signal reflections. Each extended CAN frame contains the following fields (Figure 3.4):

- **SOF:** Identifies the beginning of the frame transmission.
- **Arbitration field:** It consists the 29 bits extended ID, which includes an 11 bits base ID along with an 18 bits extension for the extended format. Additionally, it features a Remote Transmission Request (RTR) bit, a Substitute Remote Request (SRR) bit, and an Identifier Extension (IDE) bit. This structure enhances message differentiation and control within the network.
- **Control Field:** It contains the payload length (DLC) and 2 reserved more bits for the protocol.
- **Payload:** The data to be transmitted.
- **CRC:** The check the integrity of the transmission.
- **EOF:** At the end of the frame, 7 recessive bits are placed to signal the end of data transmission.
- **IFS:** 3 bits for inter-frame space to reduce the effect of interference.

To make the protocol deterministic, the Time-Triggered CAN (TTCAN) and Flexible Time-Triggered CAN (FTTCAN) with a rate up to 1 Mb/s have been introduced. However, they need to have at least one master in the network to perform synchronization, thus violating the multi-master feature of the protocol [99]. In addition, they need higher layer and hardware implementation. The recent upgrade of CAN (CAN-FD) extends the data field up to 512 bits and the rate up to 2 Mb/s and is promising for cyber-security and functional safety applications.

3.1.3 FlexRay

FlexRay is a high-speed bus system capable of achieving data rates up to 10 Mb/s. It is designed to be deterministic and error-tolerant, utilizing two communication channels to minimize the risk of failure, which can involve 2 to 4 wires. In scenarios where failsafe functionality is not required, both channels can be combined to reach a total data rate of 20 Mb/s for each FlexRay message.

The FlexRay interface comprises a communication controller that handles framing, bus access, error detection and handling, synchronization, and other essential functions. Depending on the number of channels, it may include one or two bus drivers that facilitate signal transformation between the controller and the bus.

The physical layer of FlexRay establishes guidelines to enhance immunity against high-frequency interference and electrostatic discharge (ESD), while also minimizing noise emissions. Similar to the Controller Area Network (CAN), communication relies on the transmission of differential voltages through two lines: Bus Plus (BP) and Bus Minus (BM). Twisted-pair cables are employed to mitigate electromagnetic interference, and these are typically non-shielded to reduce costs.

To prevent signal reflections, proper termination is essential. FlexRay specifies a load impedance between $40\ \Omega$ and $55\ \Omega$, meaning that the values for bus termination resistors should fall between $80\ \Omega$ and $110\ \Omega$.

The ends of the FlexRay bus can be terminated using a split bus termination, which functions as a low-pass filter. This design effectively shunts high-frequency signals to ground without disrupting the DC voltage relationships. The incorporation of LC suppression circuits within the FlexRay transceivers mitigates interference currents that may arise from asymmetrical circuits.

Additionally, the low-pass filter, formed by the coupling capacitor of the split termination and an interference suppression coil, directs high-frequency interference to ground. According to the Electrical Physical Layer Specification, the following parameters are prescribed for interference suppression coils: a line resistance of less than $2\ \Omega$, an inductance greater than $50\ \mu\text{H}$, and a leakage inductance of less than $1\ \mu\text{H}$.

However, a notable drawback of the LC circuit is that the combination of leakage inductance and the coupling capacitor can create an oscillating circuit. This interaction, along with the switching processes of the FlexRay transceiver, can lead to overshoots in the bus signals.

A single FlexRay message can transport a maximum payload of 254 user bytes. This protocol is primarily utilized in safety-critical embedded systems and advanced

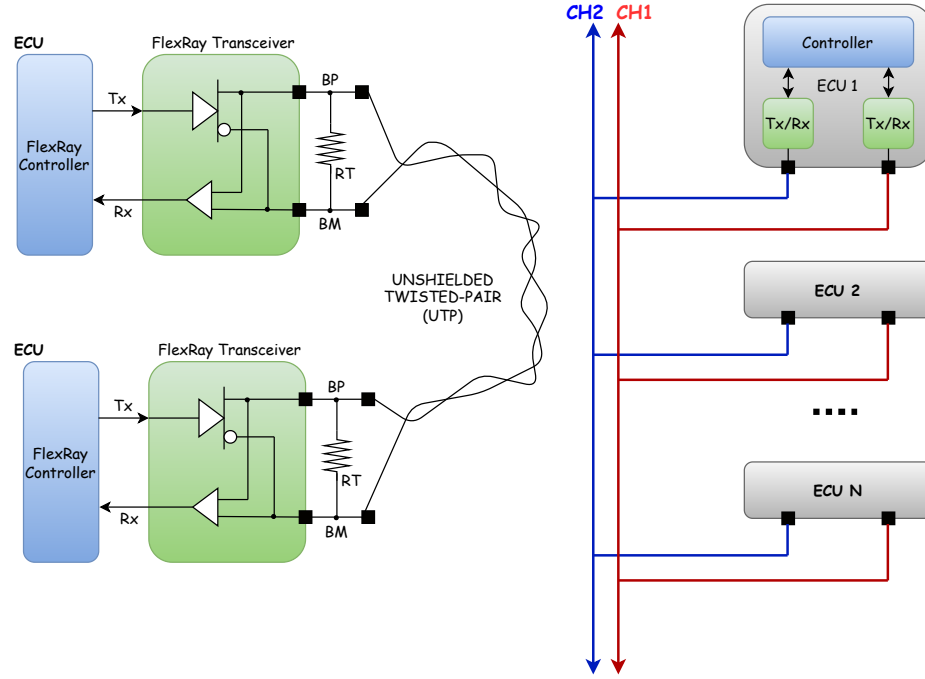


Figure 3.5: FlexRay Line Topology

control functions, characterized by its high-speed communication capabilities, making it suitable for Class D applications.

FlexRay employs Time Division Multiple Access (TDMA) for channel access, where nodes or messages are allocated fixed time slots for exclusive bus access. This structure allows for predictable timing of message transmission on the bus, necessitating synchronization among nodes.

The protocol is straightforward to configure and supports various network topologies, including line, passive-star, active-star, and point-to-point configurations. Each ECU connected to the bus must have a clock drift less than 0.15% from the reference clock.

Audi is using the Flex Ray bus system for the first time in the A8 [6]. This system allows transmission of data between the ECUs for the chassis and the assistance systems in real time. FlexRay cannot be used for multimedia and infotainment applications because it cannot provide a sufficient communication bandwidth to accommodate high-bandwidth applications [40]. An example of line topology is shown in Figure 3.5.

3.1.4 Media Oriented System Transport (MOST)

Media Oriented Systems Transport (MOST) is a high-speed multimedia networking and infotainment module fiber optic network. The protocol supports 14 Mb/s data rate (with asynchronous transmission) and 24 Mb/s (with synchronous transmission) and therefore it is used for Class D SAE applications (versions with higher data rates extending further up to 50 Mb/s and 150 Mb/s are under development or implementation). Access control in synchronous transmission is achieved through Time Division Multiplexing (TDM) while access control in asynchronous transmission is performed through Carrier Sense Multiple Access - Collision Avoidance (CSMA-CA). MOST supports a maximum of 64 devices on the bus, which has three channels of communication for different types of data transfer: Control, status and multimedia. A MOST bus system is a ring topology where each device is connected to the bus through the receiver and transmitter terminals (Figure 3.6).

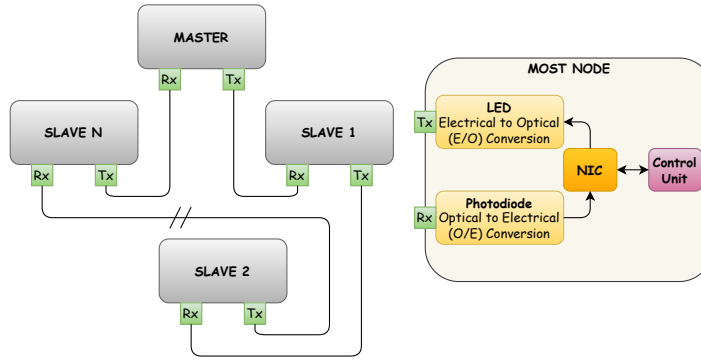


Figure 3.6: MOST Topology

There is one master which controls the synchronization of the other nodes through generation of the data frames for data transfer. Each device connected in the ring has a bypass functionality through which a device can forward the signal from the receive to the transmit terminal. This is useful in case any device is taking more initialization time or if the unit has had a shut off due to some temperature dependent fault. The nodes connected to the MOST bus integrate the interface controller and the converters to transform the electrical data inside the device into an optical signal and vice-versa.

The MOST communication system primarily uses optical signals and isolated optical fiber cables for connections between transmitters and receivers. This setup offers benefits such as immunity to electromagnetic interference, reduced emissions affecting the vehicle environment (crucial for high-rate digital signals), and lighter weight compared to shielded copper cables. However, shielded copper cables are more cost-effective and

can be enhanced with extra shielding to prevent electromagnetic interference. Recent MOST standards are moving towards standardizing the use of shielded copper cables under specific conditions.

3.1.5 Low-Voltage Differential Signaling (LVDS)

Low-Voltage Differential Signaling (LVDS) is a high-speed signaling differential interface which defines the Electrical layer only (Receiver and Transmitter) and uses twisted pair copper cables. LVDS is designed with an output voltage swing of 350 mV with speeds at better than 400 Mbps into a 100 Ω load, across a distance of about 10 m. As with all buses, the type of cable determines cable length (or the bus speed). Ribbon cable may be used for sub meter runs. LVDS Edge rates are 1 V/nS, Output voltage is 350 mV (50 mV min, 450 mV max). The center voltage is 1.2 V. LVDS uses a current-mode driver output from a 3.5 mA current source. This drives a differential line that is terminated by a 100 Ω resistor, generating about 350 mV across the receiver. The +350 mV voltage swing is centered on a 1.2 V offset voltage. While not explicitly developed for automotive applications, the high bandwidth made possible by LVDS (up to 655 Mb/s) has made LVDS an attractive option for automotive camera manufacturers [92]. It is often supported by automotive-grade. However, it has been superseded by Ethernet-based devices in recent years.

3.1.6 Power Line Communication Interface (PLC)

Power-Line Communication (PLC) is a communication technique to transmit/receive data via power lines [32] eliminating duplication wiring of power line and communication bus (Figure 3.7).

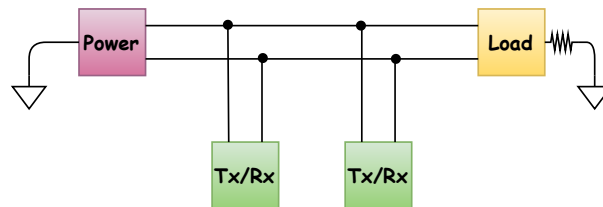


Figure 3.7: PLC Topology

In PLC systems, the communication is in general based on carrier modulations (like GMSK or OFDM) in combination with forward error correction techniques (like Turbo Coding or LDPC) to counteract the noisy channel. Since it introduce unacceptable

latency in the communication because of the carrier recovery and signal processing for message data extraction, in the literature different solutions are proposed to change LIN phy (dividing the frequency band 22-30 MHz to match EM requirements and to achieve redundancy) [52] or CAN phy (realizing transmitter with switched capacitors technique) [17] with a base-band PLC to benefit from the advantages of this communication technique. Power line communication PLC is also commonly used to extend Ethernet capabilities using the already existent power infrastructure in a building. Similarly, it can be used to extend existing protocol capabilities in a vehicle. It combines communication and power cables, which means large savings in space required to route cables through the vehicle (wiring load is reduced). For certain applications where cabling space and cost must be reduced, a PLC network could potentially extend an Ethernet network via powerlines. In addition, reducing the wiring load could also significantly decrease the weight of automobile [32].

3.1.7 Automotive Ethernet

In the automotive field Ethernet is used for driver-less applications, ADAS camera based systems, diagnosis, multimedia, and infotainment systems. It uses CSMA-CD to access the channel. Differently from other BUS technology, automotive Ethernet is a point to point networking system in which one node (for example a sensor/actuator) or an ECU is connected with another unit. To establish a connection among multiple nodes, a switch is needed in the system, which enables multiple ECUs to communicate together and forward the traffic to different nodes in the network through their physical address. Most of the Ethernet applications used in automotive use star topology (all the nodes or ECUs are connected to a central switch), even if safety-critical applications use ring topology. Ethernet provides high flexibility, high data transmission rate (supports from 100 Mb/s to 10 Gb/s), low latency, ease of installation, large data size and high flexibility. Therefore, it is also used to guarantee functional safety. Since each packet contains source and destination address, there is no need to broadcast each message like in CAN networks. It is also a protocol suitable to interconnect heterogeneous networks through automotive gateways, even if a strong synchronization to compensate clock drifts is needed in the in-vehicle network systems in order to ensure a high QoS level of the routing messages [40]. In particular, both Time-Triggered Ethernet (TTEthernet) and Audio Video Bridging (AVB) standards have been assessed to meet automotive communication requirements [2, 113]. In the following sections, we will delve deeper into their definitions.

Time-Triggered Ethernet

Time-Triggered Ethernet (TTEthernet - SAE AS6802) is designed for mixed-critical requirements by implementing a Time Division Multiple Access (TDMA) multiplexing strategy. It distinguishes between three types of flows, time-triggered flows (TT), constrained flows (RC), and Best Effort (BE), which share the same physical link [108].

Specifically, TT flows have the highest priority and are scheduled based on an offline table designed to support global synchronization. Thanks to the pre-designed slot allocation, TT flows can avoid conflicts across the entire network and are suitable for safety-critical applications due to deterministic transmission and predictable delays.

RC (Rate-Constrained) messages have a lower priority compared to TT (Time-Triggered) messages and are suitable for applications with less stringent deterministic and real-time requirements. RC communication is defined by the minimum allowed time interval between consecutive frames, known as the Bandwidth Allocation Gap (BAG), and the maximum allowed frame size. When different controllers simultaneously send multiple RC messages to the same receiver, these messages are queued in the switching machine. This queuing can lead to increased transmission jitter due to the varying arrival times of the messages.

Finally, BE (Best-Effort) messages have the lowest priority and are transmitted using the leftover network bandwidth. As a result, their performance cannot be guaranteed.

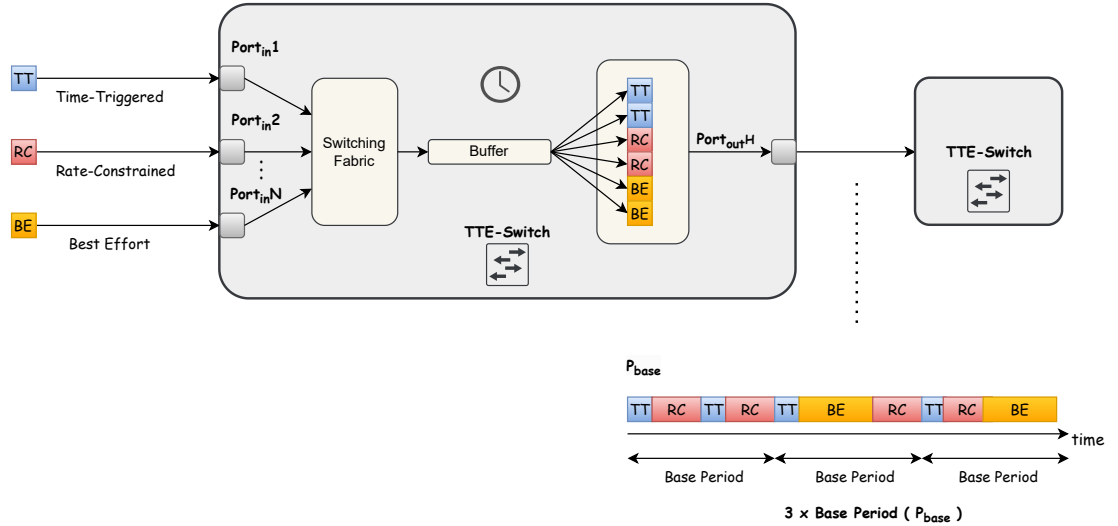


Figure 3.8: TT-Ethernet Switch Architecture and Frame Transmission

In Figure 3.8 a logical TT-Switch architecture and frame transmission concept are shown. A Time-Triggered (TT) switch processes traffic from various network nodes,

categorizing it into TT, Rate-Constrained (RC), and Best-Effort (BE) traffic based on priority levels. The TTEthernet switch ensures that TT frames are dispatched from an internal buffer precisely at the scheduled time, adhering to the predefined transmission schedule. Critical traffic is managed according to its allocated transmission time slots. Protocol Control Frames (PCFs) are specialized TTEthernet frames utilized for communication between TTEthernet components to establish and maintain synchronization across the network.

Time-Triggered Ethernet has been implemented in several space, industrial and automotive projects and components.

Audio Video Bridging (AVB)

Initially developed by the Institute of Electrical and Electronics Engineers (IEEE), Audio Video Bridging (AVB) is a collection of standards designed to ensure reliability, low latency, and enhanced synchronization for Ethernet-based switched networks, therefore enabling a real-time networking solution that allows audio/video traffic to be transported with guaranteed maximum latency [10]. In November 2012, the Audio Video Bridging Task Group was renamed to the Time-Sensitive Networking (TSN) Task Group to better reflect advancements in the field. These advancements pertain to specifications for time-synchronized, low-latency streaming services over IEEE 802 networks. Key features, for example, include:

- **IEEE 802.1AS:** Timing and Synchronization for Time-Sensitive Applications (gPTP). The Precision Time Protocol (PTP) specifies clock synchronization mechanisms to achieve high precision and accuracy over Local Area Networks (LANs), ensuring that it is suitable for TSN applications.
- **IEEE 802.1Qav:** Forwarding and Queuing for Time-Sensitive Streams. Traffic shaping is defined using priority classes that employ a straightforward credit-based queuing strategy. This approach ensures efficient bandwidth management by prioritizing data flow and maintaining network performance.
- **IEEE 802.1Qat:** Stream Reservation Protocol (SRP).
- **IEEE 802.1BA:** Audio Video Bridging (AVB) systems.

Additionally, IEEE 802.1Qat (SRP) and IEEE 802.1Qav (Forwarding and Queuing Enhancements for Time-Sensitive Streams) have been integrated into IEEE 802.1Q. In IEEE 802.1Qav, two main stream classes, Class A and Class B, are defined using specific

bits. Transmission is permitted when the credit is non-negative. During transmission, the credit decreases at a rate known as the *sendSlope*:

$$sendSlope = idleSlope - portTransmitRate \quad (3.1)$$

Accordingly, the credit increases at an *idleSlope* rate when frames are waiting to be transmitted while other frames are in the queue:

$$idleSlope = \frac{reservedBytes}{divFrameIntervalTime} \quad (3.2)$$

Therefore, the *idleSlope* is practically the queue reserved bandwidth by the bridge, while the *sendSlope* is transmission rate of the Medium Access Control (MAC) port.

The credit increases until zero is reached with a *idleSlope* rate. When it is not possible to transmit an AVB frame, the credit accumulates with a *idleSlope* rate, but in this case a positive credit is allowed. An example of Qav traffic shaping is shown in Figure 3.9.

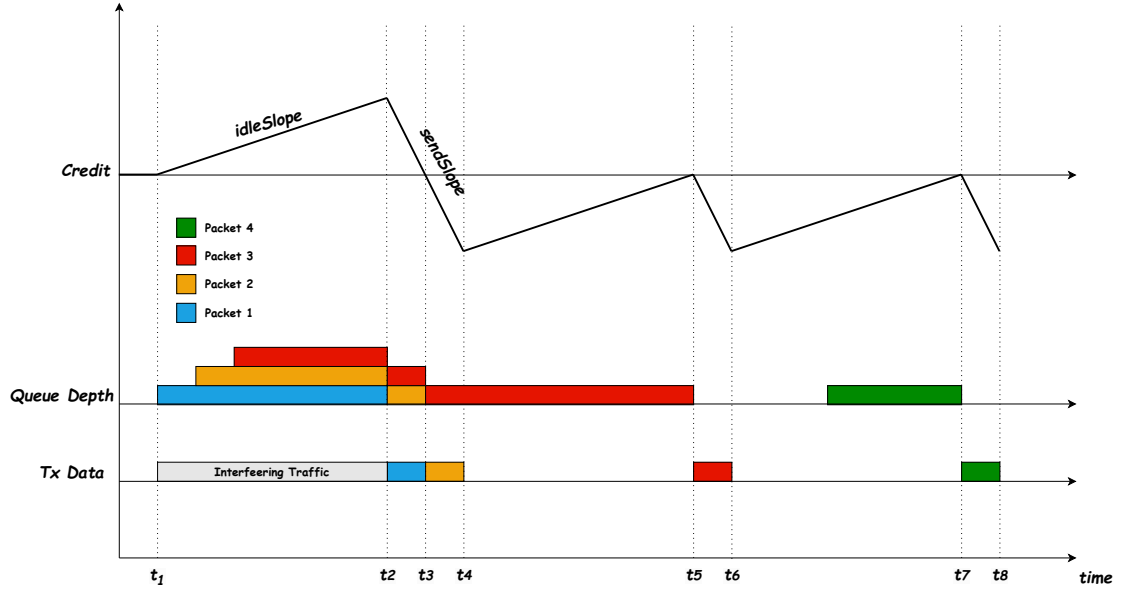


Figure 3.9: IEEE 802.1Qav Credit Base Shaping

In the example, at time t_1 there are three packets in the queue, and since the credit is positive, their transmission is postponed until the interfering traffic has finished. At time t_2 , packet 1 is transmitted and at time t_3 , packet 2 is transmitted. At time t_4 , since the credit has become negative, packet 3 is held back. At time t_5 , credit is positive and therefore packet 3 is transmitted. Between time t_6 and t_7 , packet 4 is held in the

queue and then it is transmitted until the credit becomes positive. The A and B classes are defined by the standard as follows [70]:

- **AVB Class A Traffic:** 2 ms guaranteed latency and transmission period of 125 μ s, within 7 hops.
- **AVB Class B Traffic:** 50 ms guaranteed latency and transmission period of 250 μ s, within 7 hops.

The standard recommends that at maximum the 75% of the bandwidth is reserved for AVB streams, while the remaining 25% is used for best-effort traffic. A conceptual scheme of the transmission selection algorithm is shown in Figure 3.10. The Credit Based Shaper (CBS) control the transmission of AVB frames managing their traffic priorities, thus providing an implicitly lower and upper bound limit to the data rate, as well as defining the bursting behavior of the AVB data.

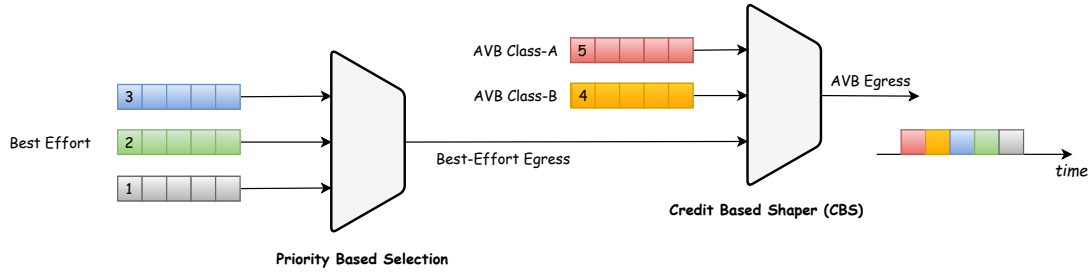


Figure 3.10: IEEE 802.1Qav Credit Base Shaping Algorithm

The IEEE 802.1Qat Stream Reservation Protocol (SRP) is used to manage admission control by considering the resource requirements of data flows and the available network resources. This ensures that Quality of Service (QoS) requirements are met throughout the entire flow path. SRP reserves resources and advertises streams from the sender (talker) to the receiver (listener). Each traffic stream is identified and registered using a 64 bit Stream-Identifier (Stream-ID), which consists of a 48 bit MAC address and a 16 bit unique ID. An example of the SRP mechanism is illustrated in Figure 3.11.

The sequence followed by the SRP protocol is:

1. The talker advertises a stream.
2. Paths along the data flow are registered.
3. Worst-case latency is calculated.

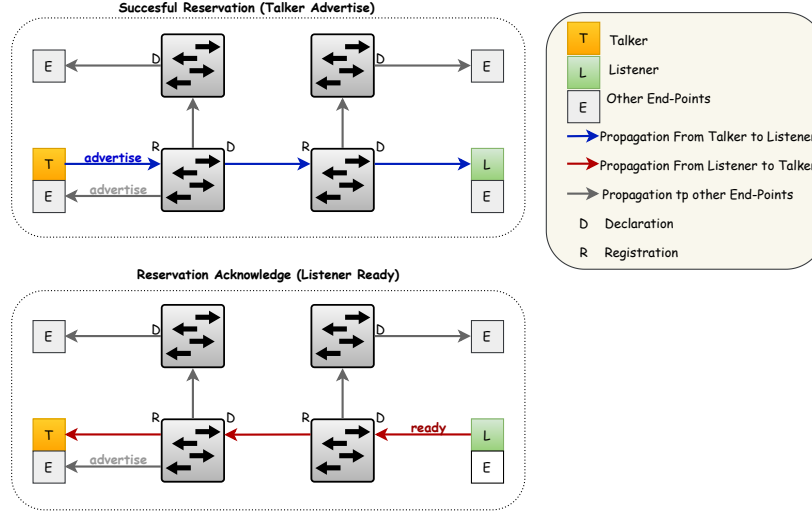


Figure 3.11: SRP Mechanism

4. AVB domain creation.
5. Bandwidth reservation

An End-to-End (E2E) mechanism is used to detect the success or failure of the reservation. If the reservation is successful, resources are allocated and configured in both the end nodes and the transit nodes along the entire data path. When a bridge can reserve the required resources, it propagates the talker advertisements to the next bridge. Otherwise, it replies with a talker failed message. When the advertisement reaches its target listener, a listener ready message is back-propagated to the talker.

To meet hard real-time requirements, TSN technology recently enhanced AVB Ethernet with additional Control Data Traffic (CDT).

3.2 Drive-by-Wire Technologies

Traditional mechanical systems, like those utilizing a steering column or direct control of the brake and accelerator by control units, are inadequate for implementing driver assistance systems and autonomous driving functionalities. These advanced systems require more sophisticated technologies to ensure precise control, safety, and reliability.

The solution is to transmit commands to the control units in digital format. This approach allows for both traditional user-driven operation and control by a unit implementing advanced algorithms, enabling seamless integration of driver assistance systems and autonomous driving functionalities.

Drive-by-Wire (DBW) systems, which replace mechanical linkages for functions like braking, steering, and throttle control, have seen widespread adoption by vehicle manufacturers in recent years. These systems collect vehicle data through sensors, which is then processed by control units. These units convert electrical signals into mechanical movements via electro-mechanical actuators. DBW systems offer numerous advantages, including improved precision, enhanced safety, and greater flexibility in vehicle design and functionality.

For instance, Throttle-by-Wire (TBW) systems can enhance engine emissions and support features like torque management, cruise control, traction control, and stability control, all integrated into the Engine Control Module (ECM) [26].

Brake-by-Wire (BBW) systems offer the benefits of reducing the vehicle overall weight and improving energy efficiency, while also minimizing vibration and mechanical power loss. Additionally, BBW systems are highly scalable, easily adaptable to various vehicle platforms, and capable of incorporating advanced features such as Automatic Braking System (ABS), Anti-Split Regulation (ASR) and Vehicle Stability Control (VSC) [51].

Finally, Steer-by-Wire (SBW) systems, which eliminate the need for a steering column, enhance engine efficiency, optimize steering response and driver comfort, and remove the necessity for hydraulic fluid in the steering mechanism [83]. Drive-by-Wire (DBW) technology is crucial for implementing advanced driver assistance systems and developing autonomous driving capabilities, as it enables mechanical actuation through both analog and digital controls.

3.3 E/E Architecture Topologies

Over the past decade, Electrical and Electronic (E/E) vehicular architectures have evolved significantly, transitioning from traditional distributed systems to domain-centralized and zonal architectures in current and future vehicles.

There are several reasons for this shift. Firstly, current distributed architectures are inadequate for accommodating higher data rates. Additionally, centralizing logic and control functions enhances overall system robustness, particularly in terms of functional safety and cyber-security. This centralization allows for more effective and secure management of these critical aspects.

The following sections will briefly discuss the main features, advantages, and drawbacks of these architectural topologies.

3.3.1 Distributed E/E Architectures

In traditional distributed architectures, embedded vehicle functionalities are spanned across numerous ECUs developed by various suppliers. Each ECU is responsible for a specific function, such as engine control, automatic gear shifting, or data visualization for the driver. The primary advantage of this approach is the lower hardware cost, as these ECUs do not require significant computational power.

Additionally, the Original Equipment Manufacturers (OEMs) primary role is system integration, ensuring that software development is supervised and controlled. The main tasks in system integration include verifying the availability of bus bandwidth and ensuring that timing constraints on communication messages are met. [19]. If an ECU fails, it can be replaced without incurring substantial costs or losing significant vehicle functionalities. This also simplifies the production line, allowing electronic issues to be resolved without high costs [75]. Figure 3.12 shows an example of distributed architecture.

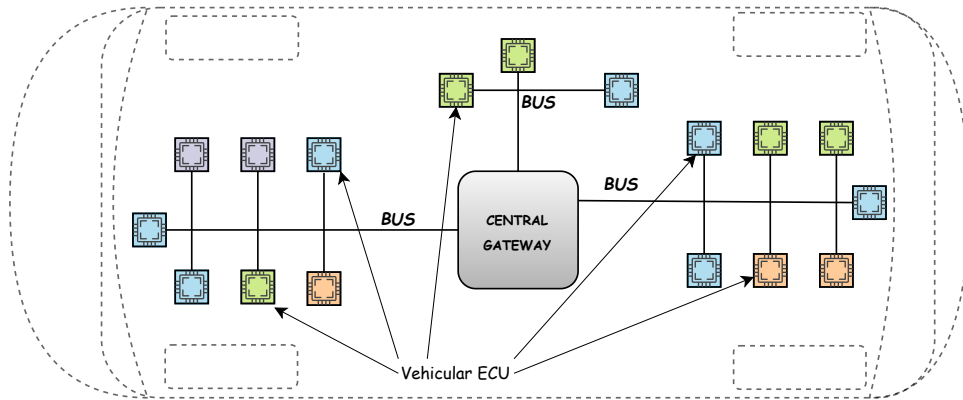


Figure 3.12: Distributed E/E Architecture

However, this architecture has several drawbacks:

- **System Complexity:** As the number of ECUs increases, the system becomes more complex and difficult to scale, leading to potential communication interference between ECUs and their integrated software components.
- **Data Management:** The growing amount of data to be managed can become difficult to handle in these architectures, necessitating high data rate communication.
- **Safety Assessments:** Performing safety assessments is complicated, as all interactions within the network among the nodes must be considered.

3.3.2 Domain-Centralized E/E Architectures

To enhance the scalability of E/E architectures, Domain-Centralized E/E systems have been introduced. These architectures cluster electronic control units (ECUs), software components, and cabling according to vehicle function domains such as powertrain, body, chassis, and comfort [28]. Each domain features a Domain Control Unit (DCU) and potentially several sub-domain ECU (I/O slaves), which primarily handle simple Input/Output (I/O) operations like sensor acquisition and actuator actuation. The DCU manages the control logic and oversees its sub-domain ECUs, with communication limited to within the domain. Typically, each DCU is a robust ECU capable of hosting various vehicle functionalities of significant complexity. DCUs communicate with each other via a high-speed link through a central gateway responsible for routing information. An example of domain-centralized architecture is shown in Figure 3.13

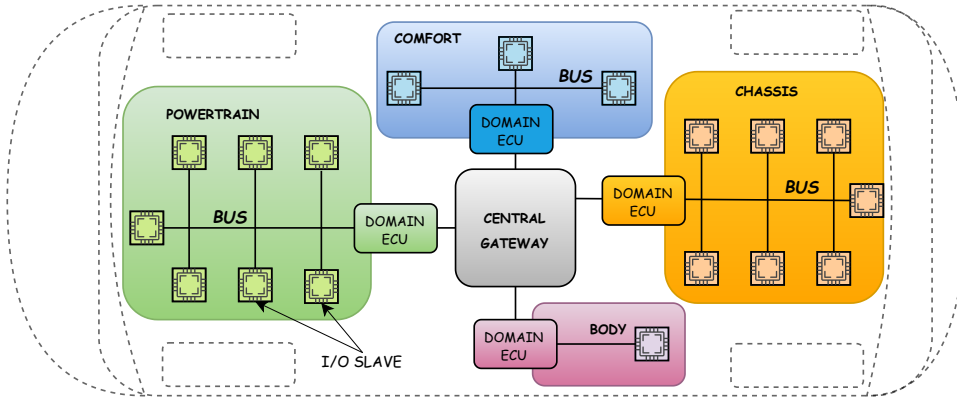


Figure 3.13: Domain-Centralized E/E Architecture

While domain-oriented architectures resolve many issues of decentralized architectures, they face two main challenges:

- **High-Performance Functions:** Advanced Driver Assistance Systems (ADAS) and similar high-performance functions require strong interaction among DCUs, potentially blurring the boundaries between them and risking a return to decentralized functions.
- **Communication Bandwidth:** As high-end functions proliferate, current communication bandwidths may become insufficient to handle the data required for future applications in E/E architectures.
- **Wiring Load:** Wiring load drastically increases as individual vehicular functions become more complex and more dependent on other functions.

3.3.3 Zonal-Centralized E/E Architectures

To address the limitations of domain-centralized architectures, zonal-centralized architectures have been proposed. In this approach, the vehicle is divided into distinct zones, such as 'front', 'center-left', 'center-right', and 'rear', based on design requirements. Each zone is equipped with a Zone Control Unit (ZCU). Unlike domain-centralized architectures, ZCUs in zonal-centralized architectures are connected to sensors and actuators (or I/O slaves) but do not perform any data processing for vehicular functions. Instead, they collect and forward data to one or more central High-Performance Computing Units (HPCUs), which handle complex processing tasks. Consequently, ZCUs can function merely as Ethernet switches if Ethernet is used as the communication medium. An example of zonal-centralized architecture is shown in Figure 3.14.

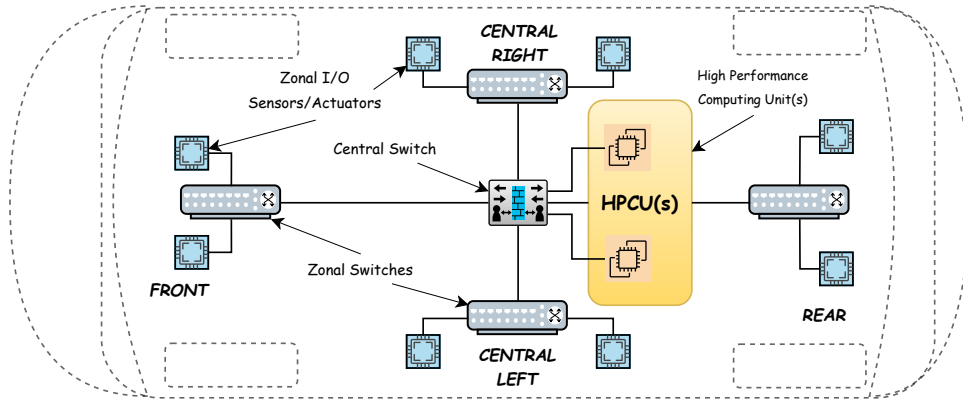


Figure 3.14: Zonal-Centralized E/E Architecture

This architectural solution can significantly reduce the vehicle's wiring load. For instance, Bosch reported a 15-20% reduction in wiring harness weight when utilizing a zonal-centralized architecture compared to a domain-centralized one [7]. Considering scalability, the integration of new technologies and systems is facilitated by the zonal clustering, which allows modifications to be made without significantly impacting the overall vehicular architecture. Regarding reliability, managing vehicle safety is simplified due to the reduced number of connections, which drastically minimizes potential points of failure. The presence of a centralized computing unit optimizes resource allocation and utilization. Finally, this architecture is well-suited to accommodate future vehicular applications such as Advanced Driver Assistance Systems (ADAS) and autonomous driving, as it can support high data rate links.

3.4 Simulation Tools

From an industrial perspective, simulation is crucial for evaluating whether different design parameters of the E/E network are satisfied. Specifically, it allows us to check network load, latency, jitter, and the correct flow of streams within the network. In this section, we will analyze various simulators and assess their capabilities for the E/E vehicular architectures of our interest. One of our objectives is to find a simulation environment that can be integrated into our tool and allows us to simulate mixed-critical in-vehicle networks (such as Controller Area Network, Ethernet, etc.).

OPNET has been used to simulate both Ethernet and CAN-based networks. Bani-abdelghany et al. proposed a simulation environment using OPNET, which extends the Precision Time Protocol (IEEE 802.1AS) to improve synchronization in hybrid networks that include both wireless and wired components [8]. This approach can be applied to industrial and vehicular systems. Additionally, Pahlevan et al. introduced a simulation framework based on the dynamic configuration mechanism of SRP, implemented on the OPNET platform [68]. This platform was also employed to evaluate CAN network performance metrics, such as end-to-end delay and throughput, demonstrating that OPNET can serve as a foundation for modeling other field-bus protocols [29]. OPNET has two main drawbacks: its high cost as a commercial tool and its limited flexibility. While it offers many models, it can be less adaptable than other simulation tools, which might restrict customization for specific use cases.

In the automotive industry, CANoe (by Vector Informatik GmbH) is a popular commercial tool that enables real-time cluster simulations of field-buses. CANoe provides a common time base for multi-bus, multi-network, and multi-system analysis purposes. It facilitates drawing conclusions regarding the chronological measurement sequence as all data and information in CANoe is time-stamped on this basis. Hardware and software time synchronization mechanisms are in place [25]. It allows the simulation of different communication standards, including CAN, FlexRay, LIN, MOST and Ethernet based networks like Internet Protocol (IP), Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Audio Video Bridging (AVB) and Time-Triggered Ethernet (TTEthernet). The Communication Access Programming Language (CAPL) language enables the creation of test sequences and provides access to various layers of the protocol stack. However, CANoe has several disadvantages. Firstly, the learning curve is steep, necessitating considerable time for developers to acquire the skills required for effective simulation. Additionally, CANoe is a costly tool, which can be a significant barrier for small companies or projects with limited budgets. Simulations can be

computationally intensive, resulting in long simulation times for large scenarios. Furthermore, there is limited customization for non-automotive protocols, which restricts its extensibility. Integration with other tools can also be challenging, often requiring extensive customization and configuration.

OMNeT++ is a widely adopted simulation framework in both research and academia, particularly known for its discrete-event simulation capabilities. It supports a variety of extensions, enabling detailed simulations of mixed-critical in-vehicle networks. This flexibility has made OMNeT++ a preferred tool for numerous studies focused on automotive Electrical and Electronic (E/E) networks, allowing researchers to model and analyze complex network behaviors and interactions with high precision.

Luo et. al. proposed a methodology to address the delay constraints of an automotive Time-Sensitive Networking (TSN) system, evaluating performance under various mechanisms such as Time-Aware Shaper (TAS), Credit Based Shaper (CBS), and Cyclic Queuing and Forwarding (CQF) [53]. Patti et. al. conducted a comparative performance analysis of three different configurations for an IEEE 802.1Q network, focusing on parameters like jitter and End-to-End (E2E) delay [70]. Using the same framework, Alderisi et. al. performed a simulation-based assessment of both Audio Video Bridging (AVB) and Time-Triggered Ethernet (TTEthernet) for Advanced Driver Assistance Systems (ADAS) and infotainment systems. This study delved into the behavior of these technologies under different workload conditions [2]. Nasrallah et al. conducted an in-depth analysis of the Time-Aware Shaper (TAS) feature in Time-Sensitive Networking (TSN), focusing on packet losses and delays within a ring network topology. Additionally, they evaluated the performance of the Asynchronous Traffic Shaper (ATS) under various traffic conditions, including sporadic, periodic, and high traffic scenarios, demonstrating its effectiveness with appropriate configurations [61].

Although these studies primarily focus on Ethernet technologies and do not consider the simulation of mixed-critical in-vehicle networks, the CoRE-RG group ¹ developed a framework based on OMNeT++ which offers the capability to model and simulate mixed CAN, FlexRay, and Ethernet vehicle networks. It also supports the implementation of gateways to facilitate data exchange between these different domains [57, 58].

The simulation framework is structured as shown in Figure 3.15.

FiCo4OMNeT facilitates fieldbus communication, encompassing both CAN and FlexRay protocols. CoRE4INET extends the INET Framework to support event-based simulation of real-time Ethernet technologies within the OMNeT++ simulation

¹<https://github.com/CoRE-RG>

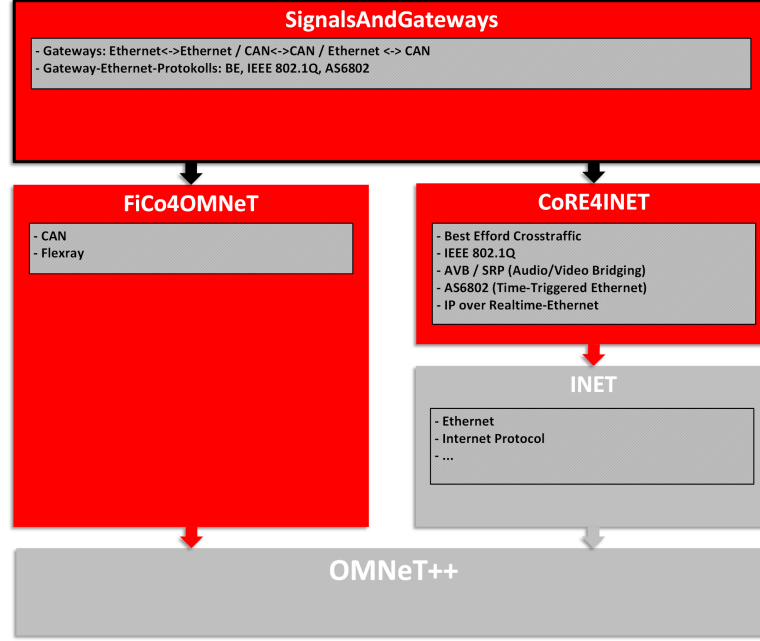


Figure 3.15: SignalsAndGateways Framework [67]

environment, including TTEthernet (AS6802) and IEEE 802.1 Audio Video Bridging (AVB). The SignalsAndGateways framework integrates CoRE4INET, INET 3.6.6, FiCo4OMNeT, and OMNeT++ 5.5.1, enabling the simulation of heterogeneous networks within OMNeT++. This framework includes gateway components that facilitate communication between Ethernet and bus technologies, allowing for comprehensive and realistic network simulations. [89]. In addition, it enables the evaluation of various network metrics, including delay, Controller Area Network (CAN) aggregation effects, and bus load, which are crucial for future in-vehicle applications. It also supports the integration of different communication mechanisms, such as combining time-triggered traffic with the Credit Based Shaper (CBS) of Audio Video Bridging (AVB).

3.5 Cutting-Edge Technologies Selection

To determine the appropriate standards for integrating a specific number of sensors into the E/E architecture, we need to establish the minimum bandwidth capabilities required. As a reference, Table 3.2 provides indicative values for the number of sensors and their rates in semi-autonomous vehicles [46].

In Table 3.3, key features of the communication standards previously discussed are summarized. Notably, the Ethernet-based protocols (such as AVB/TSN and TTEthernet) emerge as the most suitable options for integrating new, high-data-rate demanding

Table 3.2: Requirements for Semi-Autonomous Vehicles [46]

Sensor Type	Number of Sensors	Data Rate per Sensor
RADAR	4 - 7	0.1 - 15 Mbps
LiDAR	1 - 5	20 -100 Mbps
Camera (Uncompressed)	6 - 12	500 - 3500 Mbps
Ultrasonic	6 - 8	<0.1 Mbps
Motion, GNSS, IMU	20+	<0.1 Mbps

features.

While Low-Voltage Differential Signaling (LVDS) remains a viable choice, it is better suited for point-to-point connections directly to an Electronic Control Unit (ECU). In contrast, Ethernet can efficiently handle diverse data streams within a switched network, thereby reducing the overall cable length required.

Table 3.3: Comparison of Technologies for In-Vehicle Networks

	LIN	CAN	CAN-FD	FlexRay	Most	LVDS	AVB/TSN	TTEthernet
Rate	up to 20 kb/s	up to 1 Mb/s	up to 2 Mb/s	up to 10 Mb/s (or to 20 Mb/s)	up to 14 Mb/s (or to 24 Mb/s)	up to 655 Mb/s	from 100 Mb/s to 10 Gb/s	from 100 Mb/s to 10 Gb/s
Medium	Single Wire	Two Twisted Wire	Two Twisted Wire	2 to 4 twisted/optical fibre wires (2 CH)	Optical fibre	Two Twisted Wire	Optical fibre	Optical fibre
Access Scheme	Serial, Master-Slave	CSMA-CD, Multi-Master	CSMA-CD, Multi-Master	TDMA	TDM (sync), CSMA/CA (async)	Not Defined	CSMA/CD, TDMA	CSMA/CD, TDMA
Frame Length	1,2,4,8 Byte	8 Byte	64 Byte	254 Byte	up to 384 Byte	Higher protocol dependent	1518 Byte	1518 Byte
Max Nodes	16 (7 in practice)	32 ECU, 253 CA	32 ECU, 253 CA	NS	64	1	Not Defined	Not Defined
Topology	Bus	Bus	Bus	Line, Passive star, Active-star, P2P	Ring	P2P	P2P Star, Ring, Switched Network	P2P Star, Ring, Switched Network
Applications	Class A	Class B, C	Class C	Class C,D	Class D	Class D	Class D	Class D

Regarding system topologies, it is evident that zone-based architectures are becoming the cornerstone of contemporary and future automotive designs. These architectures offer significant advantages, such as reducing wiring load, facilitating system integration, enhancing safety, and supporting high data rate links. Additionally, they enable and facilitate the integration of Drive-by-Wire (DBW) technologies.

Building on the current state-of-the-art and previously discussed Design Space Exploration (DSE) techniques, as well as the background related to automotive E/E networks, we propose a novel approach tailored to the Off-Road Vehicle (ORV) domain. This approach addresses critical challenges related to regulation requirements and the expansion of E/E architectures to support autonomous driving capabilities through a combined DSE technique that integrates optimization and simulation. Specifically, we

incorporate advanced technologies and concepts for designing future vehicular networks, including Drive-by-Wire systems, Ethernet standards, and zone-based architectures.

As anticipated, we face the requirement and engineering challenge of maintaining the existing CAN-based components unchanged. Commercially, a complete network redesign is not feasible due to time constraints, risk, and development costs. However, we will apply new technological concepts, such as zone-based topology, to the new sections that need to meet regulation and autonomous driving functionalities. To accurately simulate the entire system, we need to use gateways that exchange data between the low data rate (CAN) and high data rate (Ethernet) domains. Consequently, the CoRE framework is the most suitable simulation tool for our purposes [67].

Although we are not dealing with a fully centralized architecture, we will explain in the conclusion how a smooth transition towards a fully zonal architecture is possible. This approach provides car designers with the flexibility to gradually integrate functionalities from the low data rate domain into the high data rate domain over time, thereby benefiting from new technological solutions without incurring risky or costly choices during the design phase.

Chapter 4

Methodology

In this chapter, we describe our design exploration methodology. We begin with an overview of the method, followed by a detailed explanation of its main components. This includes the mathematical formulation of the optimization problem and the strategy for reformulating constraints. We revisit the concepts introduced in ArchEx and enhance the formulation by incorporating the necessary adaptations [41].

4.1 Combining Optimization and Simulation

As anticipated in the previous chapters, in our exploration methodology we combine optimization and simulation to benefit from the advantages of both approaches. In Figure 4.1 a simple sketch of the main steps of the technique is shown.

The initial step involves defining the system specifications, which encompass both architectural constraints and quantitative constraints. Architectural constraints pertain to the logical system topology, specifying the number and types of components present in the network architecture and their connectivity. Quantitative constraints relate to the system real implementation, addressing factors such as power consumption, cost, and weight. These specifications, along with a cost function definition, are encoded using a Mixed-Integer Linear Programming (MILP) method. This method outputs a proposed system architecture, detailing the system topology and its implementation, which includes an association between the components in the system topology and the real devices that perform the specified functions.

To verify the proposed solution, the optimized architecture is automatically modeled through simulation files, which also integrate the fixed design part related to the CAN bus network. A simulation of the obtained architecture, integrated into the complete

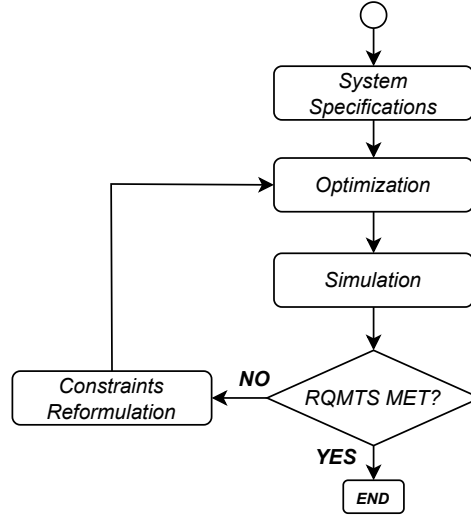


Figure 4.1: Architectural Exploration Methodology

system, is then performed to ensure that the design specifications are met. If they are not met, a gradient-ascent-based methodology is applied to reformulate the problem constraints, thereby speeding up convergence. This iterative process continues until the design specifications are satisfactorily met.

Through optimization, we aim to explore the design space and provide candidate solutions, thereby avoiding repetitive trial and error during the design phase. The simulator offers precise results regarding the integration of the candidate solution into the entire system architecture, accurately estimating design parameters such as latency, jitter, and network load. We then refine our design solution by reformulating the input requirements to the optimizer until the design specifications are met. This approach allows us to maximize the benefits of both optimization and simulation. An more detailed overview of the main advantages and disadvantages of optimization and simulation is reported in Table4.1.

By integrating optimization and simulation, we leverage the strengths of both techniques while mitigating their individual weaknesses. The optimizer can efficiently prune the design space to focus on viable solutions, while the simulator provides detailed and accurate performance evaluations for these solutions. This iterative process helps in converging to an optimal architecture that balances both theoretical and practical considerations.

In the next sections, we provide details and formalization about the technique.

Table 4.1: Advantages and Disadvantages of Optimization and Simulation

	Advantages	Disadvantages
Optimization	Efficiency: Optimization algorithms can quickly narrow down a large design space to a set of feasible solutions.	Model Limitations: Relies on abstract models that may not capture all real-world complexities.
	Scalability: Capable of handling large-scale problems with many variables and constraints.	Computational Demand: Some optimization problems can be computationally expensive and time-consuming, if large number of parameters are used.
	Precision: Provides mathematically optimal solutions based on the given models and constraints.	Feasibility: May generate solutions that are theoretically optimal but practically infeasible due to unmodeled/hidden constraints.
	Automation: Can be automated to generate candidate solutions without manual intervention.	Complexity: Writing a custom software to model a custom problem could require time and expertise.
Simulation	Detail-Oriented: Capable of modeling and analyzing detailed system behaviors, such as protocol handshakes and runtime collisions.	Scalability Issues: Not suitable for exploring large design spaces due to high computational costs.
	Realism: Provides a more realistic assessment of system performance under various conditions.	Time-Consuming: Detailed simulations can be very time-consuming to run.
	Integration Testing: Allows for testing the integration of different standards and protocols within the system.	Limited Scope: Typically evaluates only a small subset of possible configurations, limiting the breadth of exploration.
	Flexibility: Can be used to evaluate a wide range of scenarios and configurations.	Complexity: Setting up and running simulations can be complex and require significant expertise.

4.2 Optimization Methodology

In this section, we formalize the optimization strategy by providing precise definitions, constraints (expressed as mixed-integer linear constraints), and the necessary mathematical framework to model and solve the optimization problem. Additionally, we present illustrative examples to enhance our understanding of the practical implications arising from the mathematical formulations. As anticipated, we refer to and revisit the concepts introduced in ArchEx, highlighting the most useful ones for our problem formulation [41].

4.2.1 Modeling of System Architecture and Mapping

Our system architecture, referred to as Electrical and Electronic (E/E) architecture, is represented by a graph G . This graph delineates the interconnections among components selected from a library $\mathcal{L}$, ensuring compliance with a defined set of composition rules.

Components can vary in type; for instance, a component might be a sensor (such

as a radar or a camera), a switch, or a processing ECU, like an Advanced Driving Assistance Control Unit (ADCU), also known as High-Performance Computing Unit (HPCU). Each component in the library has a set of attributes that capture specific features, such as cost, power consumption, weight, and data rate. Composition rules are used to define the permissible connections between components.

In our case, we rely on the flow of information - data acquired from sensors are collected and processed by a central ECU. Therefore, we use flows to model the problem, designating some components as sources and others as sinks.

Definition 4.2.1 (System Architecture). *A system architecture is a directed graph $G = (V, E)$, in which V is a set of components or nodes (such as sensors, actuators, ECUs, etc.) and E is a set of edges representing a connection from v_i to v_j (for example a CAN bus, Ethernet, etc.), where $i, j \in \{1, \dots, |V|\}$ with $|V|$ the cardinality of V .*

The graph G is represented through an adjacency matrix E , where edges $e_{ij} \in E$ are binary decision variables which are equal to 1 ('True') if a connection between components v_i and v_j exists, and 0 ('False') otherwise. Therefore, an edge between v_i and v_j is instantiated only if e_{ij} is equal to 1. If at least one incoming edge variable e_{ij} or one outgoing edge variable e_{ji} , then the node v_i is connected (instantiated).

For a given E/E system, the set of all possible architectures constitutes an architectural space. Designers can explore this space to identify a feasible, optimal, or sub-optimal instance that meets the application requirements.

The formal definitions of system template and system topology are formalized as follows:

Definition 4.2.2 (System Template). *A re-configurable system template, denoted with $\mathcal{T}$, is defined as a graph with a fixed set of components V and a variable set of edges E . To establish their presence or absence, each edge is associated with a binary decision variable e . In particular, e_{ij} is used to denote the connection from component v_i to component v_j . The size of the template is determined by the number of nodes in the graph, i.e. $|\mathcal{T}| = |V|$*

Definition 4.2.3 (System Configuration). *A system configuration E^* is an assignment over the variables in E , marking the subset of edges from $\mathcal{T}$ that are instantiated. E^* is an instance of $\mathcal{T}$.*

Therefore, since it is re-configurable, $\mathcal{T}$ is an input for the exploration problem. Through composition rules it is possible to restrict connection between nodes in $\mathcal{T}$.

For example, a sensor cannot be directly connected to an ADCU, it must be connected through a switch. In other words, a system configuration can be viewed as a design decision that determines which components are selected to implement the solution and how they are interconnected.

An example of architecture selection is shown in Figure 4.2.

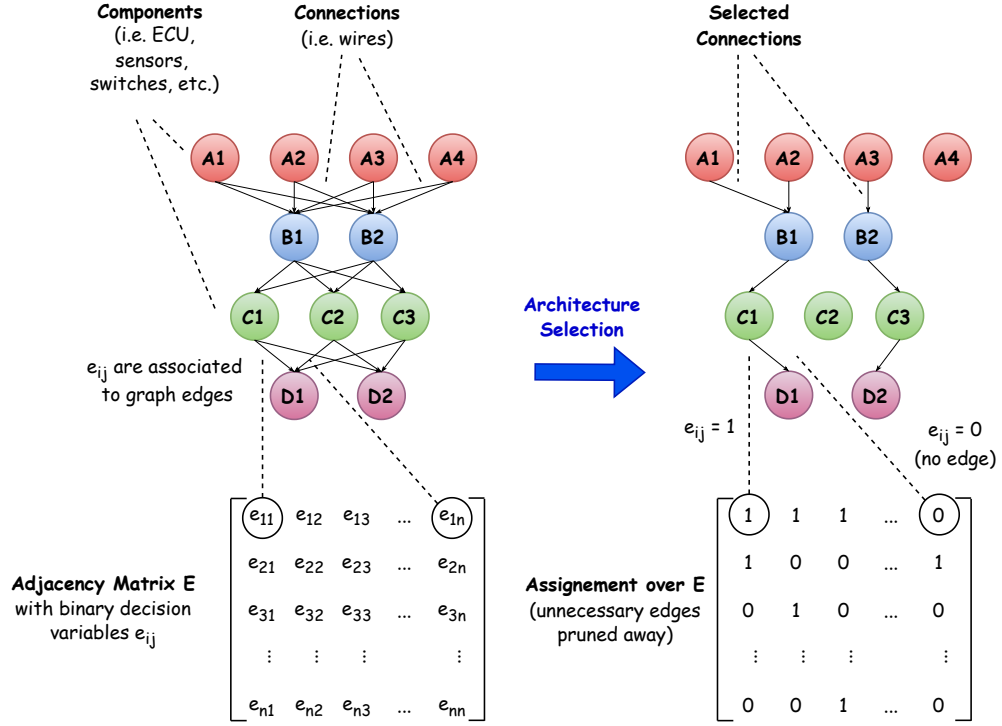


Figure 4.2: Example of System Architecture Selection

The system template is made of 4 components type having different colors, A,B,C,D. The template $\mathcal{T}$ includes all the possible connections, but the associated graph is not complete since the connections are restricted through connection rules (for example direct link between A and C are not allowed). The total number of components is 11. Each connection is represented in the adjacency matrix E through a binary variable e_{ij} which is an element of the matrix. The optimization process involves selecting an architecture by pruning unnecessary edges. Consequently, after optimization, the binary variables are set to 1 if a connection exists and 0 otherwise. Additionally, all nodes except C2 and A4 are utilized, as C2 and A4 lack both incoming and outgoing connections. Each component in the graph, referred to as a 'virtual component', can have various implementations, each with distinct properties. For instance, components

of type A could be sensors, which might be either radars or cameras, each differing in cost, weight, power consumption, and data rate. This system mapping process is defined as follows:

Definition 4.2.4 (System Mapping). *A mapping $\mathcal{M} : V \rightarrow \mathcal{L}$ is an association between 'virtual components' (graph nodes) $v \in V$ and 'real components' (physical devices in the library) $l \in \mathcal{L}$. The mapping is represented by assigning to each pair (l_i, v_j) , where $l \in \mathcal{L}$ and $v \in V$, a binary decision variable $m_{ij} \in \mathcal{M}$, where $\mathcal{M}$ is the mapping matrix and m_{ij} is equal to 1 if one v_j is mapped to l_i and 0 otherwise.*

Therefore, a component v_j is not used in the final architecture if it is not associated to any library component l_i . Consequently, the instantiated components are those that are mapped to at least one library component:

$$\sum_{i=1}^{|\mathcal{L}|} m_{ij} \geq 1 \quad (4.1)$$

Auxiliary boolean variables δ_i are used to define system requirements, where δ_i is equal to 1 if v_j is instantiated, i.e. the sum of mapping variables for v_j is greater than 0, and 0 otherwise.

In our problem formulation the edges mapping is fixed, but it can be modeled if necessary (for example a link between an ECU and an actuator could be a wire, a CAN connection, or another type of link).

In Figure 4.3 is shown a mapping example for a system architecture and a component library.

The association of a virtual component of the template $\mathcal{T}$ to real components from the library $\mathcal{L}$ is achieved through the mapping matrix $\mathcal{M}_{m \times n}$. For example component A1 is associated to an automotive camera and component C1 to an Ethernet switch. In that case, the corresponding m_{ij} element of the mapping matrix $\mathcal{M}$ is equal to 1 and the components A1 and C1 are said to be instantiated in the system architecture.

Definition 4.2.5 (Path). *A network route (path) $\pi(v_0 \rightarrow v_n)$ is a sequence of components (nodes) such that $e_i, e_{i+1} \in E$ for each i . The length of π is denoted through $|\pi|$. For a pair of components $q = (v_s, v_d)$, where v_s is the source component and v_d is the sink component, we define $\pi^q := \pi(v_s \rightarrow v_d)$.*

In order to compute the network paths, binary decision variables y_{ij}^π can be used to label the edges of the graph $e_{ij} \in E$. In particular, y_{ij}^π equals to one if e_{ij} connects the components (v_i, v_j) in π and zero otherwise. As a result, for each path π exists

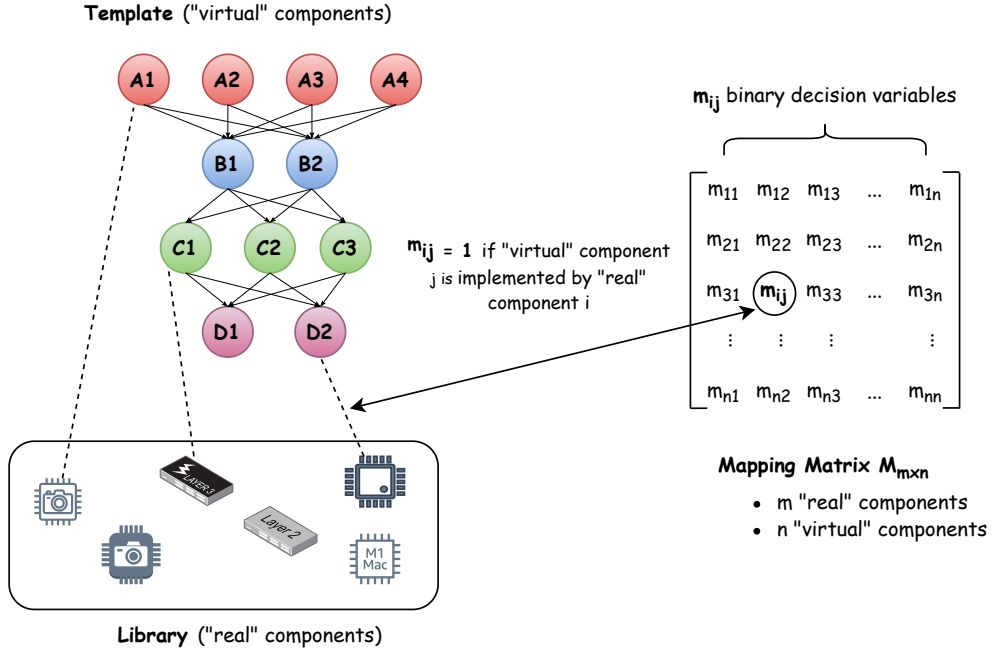


Figure 4.3: Example of Mapping

an associated vector y^π with $|y^\pi| = |E|$. In addition, the components $v \in V$ are also labeled with binary decision variables $\omega_i^\pi : \omega_i^\pi$ which equal to one if a component v_i belongs to π and zero otherwise. We call therefore ω_i^π and y^π path variables, which can be related to each other and to graph variables e_{ij} through linear constraints.

In Figure 4.4, a path definition example is shown.

The path $\pi^q = (A3, B2, C3, D2)$, where $q = (A3, D2)$. For the node in the path B3, the path variable ω_i^π is equal to zero, because the component is not in the path π^q . Similarly, for the node C3, the path variable ω_i^π is equal to one because the component is in the path π^q . Regarding the edges, the path variable y_{ij}^π is equal to one if the edge e_{ij} connects two components in π^q , and zero otherwise. Note that y_{ij}^π can be equal to zero even if the edge e_{ij} is equal to one.

Path variables are created exclusively for node pairs that need to be connected as specified by the user. Multiple paths can be defined between the same pair if more than one replica is required for example to guarantee reliability. This approach helps manage the exponential increase in problem complexity that comes with enumerating all possible paths. In addition, it is possible to associate all nodes of $\mathcal{T}$ with variables y_{ij}^π for each required path π , or in other words, any connection of $\mathcal{T}$ can be used in any path π . Practically, all possible paths are symbolically encoded in all possible π , with

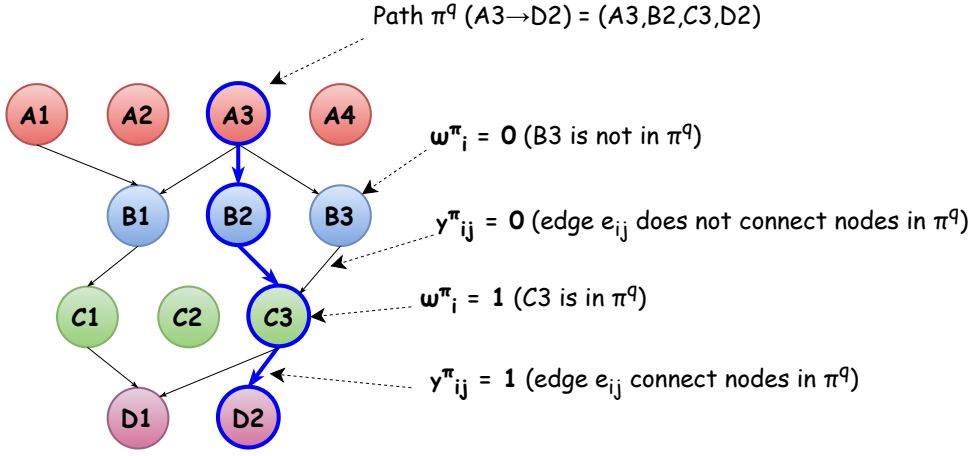


Figure 4.4: Example of Path

the corresponding complexity which is also exponential.

A set Π^q is used to store all required paths for a pair q . In general, $|\Pi^q| = 0$ if there are no paths required for a pair q , while $|\Pi^q| > 0$ if one or more paths are required for a pair q . The vectors y^{π^q} and ω^{π^q} can be grouped in a set $R = \{y^{\pi^q}, \omega^{\pi^q} | q \in Q, \pi^q \in \Pi^q\}$, Q being the subset of pairs of nodes in V for which a path is required.

Definition 4.2.6 (Routing). A routing R^* is an assignment over the set R composed of variables y^{π^q} and ω^{π^q} , marking the used connections and components of $\mathcal{T}$ which are used in system required paths. In other words, it is a routing table for the components used in the system.

We define P a partition over V , where all the components belonging to the same subset in P have the same type. We also denote with t_v the type t of a node (component). The nodes v_i and v_j with the same type t can also have different sub-types s_{v_i} and s_{v_j} .

Through library composition rules and user-defined interconnection constraints, it is possible to allow or restrict connections and paths among nodes in the graph.

If two components sub-types allow them to be connected together through a path, they are said to be compatible. A path π is valid if signal coming from the source in π can be accepted by its sink. Consequently, if a source and a sink are not compatible, intermediate components must be included in a path (for example, a sensor must be connected to a central processing unit through an Ethernet switch).

We assume an automotive E/E architecture where sensors are connected to a central processing unit via switches. Additionally, we consider a zonal topology in which a

central switch aggregates data from all zonal switches and CAN-to-Ethernet gateways, conveying it to the central processing ECU. Consequently, a valid path from a source (sensor) to a sink (processing ECU) must include two switches: a zonal switch followed by a central switch.

Let S_1 and S_n in the partition P include all the sinks and the sources in the network, respectively. Consequently, a functional link F_i is a set containing all valid paths from any source in S_1 to any source $v_i \in S_n$. These links are essential for the system correct operation. For instance, in an automotive E/E architecture, they form the data paths that signals follow.

Definition 4.2.7 (Functional Flow). *An ordered sequence of node types $(t_1, \dots, t_n)$ defines a functional flow $\mathcal{F}$ (i.e., the link between a sink and a source). It is also possible to write $v_i \leq_{\mathcal{F}}$ if v_i is precedent to v_j in $\mathcal{F}$, and $v_i \sim_{\mathcal{F}}$ if v_i and v_j have the same type.*

A path π satisfies $\mathcal{F}$ ($\pi| = \mathcal{F}$), if the ordering of component types in π does not violate the ordering in $\mathcal{F}$. Formally, we can define this statement as follows:

$$(v_i \sim v_{i+1}) \vee (v_i \leq v_{i+1}) \quad \forall i \in \mathbb{N} : 1 \leq i \leq |\pi| - 1 \quad (4.2)$$

Multiple functional flows, indicated as $\mathcal{F}_1, \dots, \mathcal{F}_n$, can be defined to represent ordering of components which realize different functions within the same system architecture. In an automotive E/E architecture, one can distinguish between sensing flows and actuation flows. In a sensing flow, data are collected from sensors and forwarded to a processing unit. For actuation flows, the processing unit generates commands that are sent to actuators to perform the required functions.

In Figure 4.5, an example of a functional link valid is shown.

In the example, it is represented a valid functional link, F_{D2} , connecting the sources $A1$ and $A2$ to $D2$. It consists of three paths: $A1, B1, C1, D2$, $A2, B2, C2, D2$ and $A2, B2, C3, D2$.

In Figure 4.6, an example of functional flows is shown.

In particular, $\pi(A1, D1)| = F(A, B, C, D)$, which means that it is a valid functional flow. On the contrary, $\pi(A4, D2)| \neq F(A, B, C, D)$, which means that it is not a valid functional flow, because of the backward connection between $C3$ and $D2$.

Given the definitions above, the architecture exploration methodology should address the selection of the system topology, including component interconnections and routing, as well as the sizing of each component by mapping it to a specific implementation from a domain-specific library. The resulting architecture must comply with a

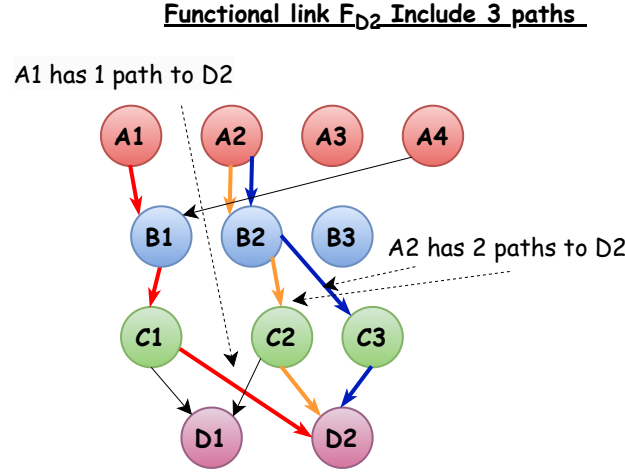


Figure 4.5: Example of Functional Link

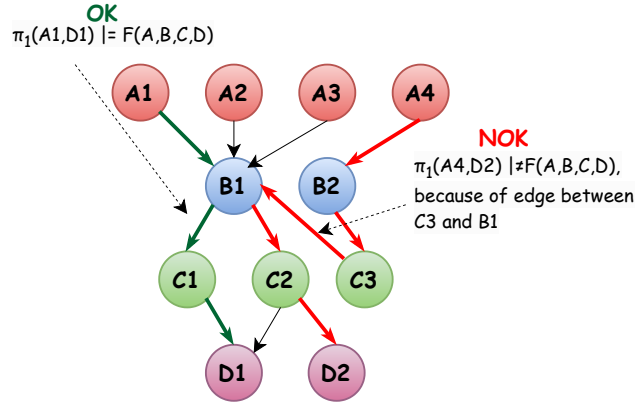


Figure 4.6: Example of Valid and Invalid Functional Flows

set of constraints. Some constraints are inherent to the structure of the given formulation, ensuring consistency, while others are application-specific requirements. Thus, all design problems based on this formulation must satisfy the structural constraints, while the application-specific constraints vary by domain and design. This approach allows us to define a common subset of variables and constraints as the foundation of the exploration problem.

In general, the use of ω^{π^q} is not enough to compute a path π^q , and therefore a routing R^* , because it does not provide information about the ordering of components in

π^q . The variables y^{π^q} , at the same time, are used to explicitly compute the connection (edges) of π^q , and hence the components can be determined through this information. Consequently, ω^{π^q} can be seen as auxiliary variables, and are in general redundant in defining the routing R . However, these variables can be used to express application requirements, ranging from timing to reliability constraints, if modeled through the optimizer. In our case, we rely more on checking timing constraints through the simulator, as the complexity of the protocols we are dealing with is very difficult to approximate with the optimizer.

4.2.2 Cost Function

Each node and edge of $\mathcal{T}$ is associated with a cost value (weight, monetary cost, power consumption, etc.). The sum of all the instantiated components (nodes) and edges (connections) costs forms a cost function:

$$\sum_{i=1}^{|V|} \delta_i c_i + \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} e_{ij} \tilde{c}_{ij} \quad (4.3)$$

where c_i is the cost of the node v_i , $\tilde{c}_{ij}$ is the cost of the edge e_{ij} , and δ_i is a binary decision variable, which is equal to one if the component is instantiated and zero otherwise. Depending on the problem type, some terms in the cost function (4.3) can be omitted. To summarize, the weighted sum of different contributions forms the overall cost function, in which particular weights can be set to zero when needed.

For instance, an objective could be to minimize the system power consumption, cost, or weight, or to maximize or minimize the bandwidth usage of the backbone, depending on the specific application goals.

4.2.3 Interconnection Constraints

Linear arithmetic constraints are used to restrict or enforce valid connections among different nodes. Let P be a partition over V , such that all components that belong to the same subset in P have also the same type. Given A , B and C sets in P , the interconnection constraints are expressed as:

$$\sum_{j=1}^{|B|} e_{a_i b_j} \geq (\leq, =) 1 \quad \forall i \in \mathbb{N} : 1 \leq i \leq |A| \quad (4.4a)$$

$$\bigvee_{i=1}^{|A|} e_{a_i b_j} \leq \bigvee_{k=1}^{|C|} e_{b_j c_k} \quad \forall j \in \mathbb{N} : 1 \leq j \leq |B| \quad (4.4b)$$

where $e_{a_i b_j}$ is a connection from component a_i to component b_j (and similarly $e_{b_j c_k}$). Constraints (4.4a) ensure that at least (at most, exactly) one connection from component $a_i \in A$ to component in $b_j \in B$ exists. Constraints (4.4b) ensure that if a component b_j has a connection to any component in A , then it shall have at least a connection to one node in C .

4.2.4 Mapping Constraints

We denote with $\mathcal{L}_k$ and P_k the subsets of $\mathcal{L}$ and V which include all the elements of type k in $\mathcal{L}$ and V . The mapping matrix for type k is defined as m^k , where $m_{ij}^k = 1$ if a virtual component (corresponding to a graph node $v_j \in P_k$) is implemented by the component $l_i^k \in \mathcal{L}_k$. Denoting with E the adjacency matrix of $\mathcal{T}$ (i.e., $e_{ij} = 1$ if a connection from node v_i to node v_j exists, and zero otherwise), type k mapping constraints can be expressed as:

$$\bigvee_{i=1}^{|\mathcal{L}_k|} m_{ij}^k = \bigvee_{i=1}^{|V|} (e_{ij} \vee e_{ji}) \quad \forall j \in \mathbb{N} : 1 \leq j \leq |P_k| \quad (4.5a)$$

$$\sum_{i=1}^{|\mathcal{L}_k|} m_{ij}^k \leq 1 \quad \forall j \in \mathbb{N} : 1 \leq j \leq |P_k| \quad (4.5b)$$

Constraints (4.5a) ensure that a mapping between real and virtual components in $\mathcal{L}$ exists for all components of type k which are instantiated, while constraints (4.5b) ensure that the virtual components are never mapped to more than one library component. This encoding approach helps the exploration of different types of implementations, since a change in $\mathcal{L}$ results in only a change in mapping constraints.

In Figure 4.7 an example of correct and incorrect mapping scenarios is shown.

For example, $A2$ is not correctly mapped because it is associated both to the elements l_1 and to l_3 from the library, while $A1$ is not correctly mapped because it is associated to none of the library components. Correct examples of mapping are instead related to $A3$, which is uniquely associated to l_2 library element, and to $C2$, which is not associated because it is not instantiated.

It is important to emphasize that this optimization process is crucial as it enables the realization of the system architecture using real off-the-shelf components. Consequently,

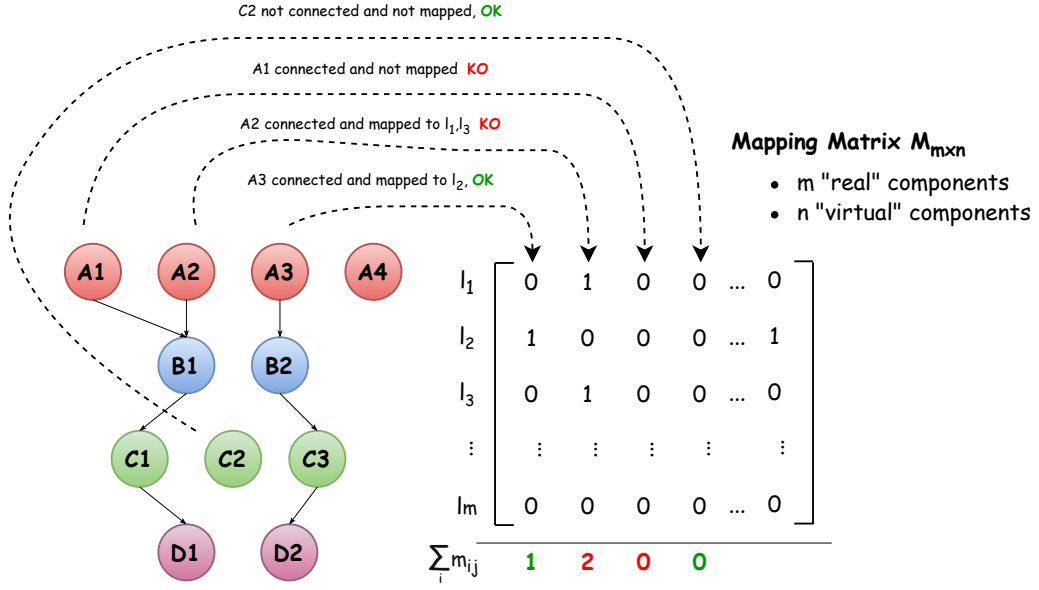


Figure 4.7: Example of Correct and Incorrect Mapping Scenarios

the outcome of the overall optimization process is not only the system topology-detailing the number of components and their interconnections-but also the system implementation. This includes specifying the exact components required to construct a functional real-world system, such as the specific models of cameras, switches, processing units, and so on.

Therefore, upon completion of the optimization process, a matrix for each type of component is generated. This matrix specifies which library components (real components) are selected to implement the virtual components.

4.2.5 Rate, Cost, Weight, and Power Consumption Constraints

In our problem formulation, we do not consider the cost of the connections, as it is negligible compared to the cost of other components used in the system. Instead, we aim to adhere to project constraints, which include the system cost, weight, power consumption, and the communication rate of the backbone.

These constraints are directly related to design requirements. For instance, the budget allocated for the vehicle E/E part is limited to a specific amount. Additionally, mechanical designers may specify weight limits for the electronic system to ensure compliance with their calculations and mechanical integration requirements. Electri-

cal teams might design a power distribution system with a constrained power budget, while the backbone link of the in-vehicle communication system must adhere to various bandwidth limits to prevent data traffic saturation.

We define these system constraints as follows:

$$\sum_{i=1}^{|V|} \delta_i c_{i,t} \leq \lambda_t, \quad t \in \{Rate, Cost, Weight, Power\ Consumption\}$$

where $c_{i,t}$ is the specific cost value t of component v_i , and λ_t is the maximum value that the specific cost value t shall be for the entire system (i.e., maximum cost, weight, power consumption, etc.).

4.2.6 Linearization Techniques

Since the methodology leverages mixed integer linear programming (MILP) constraints, all constraints and cost functions related to the problem must be linear. This means that all functions computing system properties must consist of linear operations over their arguments, which are decision variables. If a property cannot be expressed as a linear constraint, techniques must be used to linearize it, often through the use of auxiliary variables.

Generally speaking, linear operations with decision variables involve only summation and multiplication by a constant. Most non-linear operations, such as AND, OR, NOT, and XOR, can be expressed in linear form using standard techniques [106]. Additionally, we need a method to express the maximum function in terms of linear constraints. The following definitions offer a formal methodology for implementing several commonly used operations in a linear form.

Definition 4.2.8 (Logical AND Linearization). *Let us consider a logical AND operation between two boolean variables operands x_1 and x_2 . To linearize the operation, the logical AND $x_1 \wedge x_2$ shall be replaced using an auxiliary variable y , and the following constraints shall be added to associate y with the operands: $y \geq x_1 + x_2 - 1, y \leq x_1, y \leq x_2$. Given the general case of more operands $x_1, \dots, x_n$, it is possible to express the constraints as follows: $y \leq x_i \forall i \in \mathbb{N} : 1 \leq i \leq n, y \geq \sum_i^n x_i - (n - 1)$.*

Definition 4.2.9 (Logical OR Linearization). *Let us consider a logical OR operation between two boolean variables operands x_1 and x_2 . To linearize the operation, the logical OR $x_1 \vee x_2$ shall be replaced using an auxiliary variable y , and the following constraints shall be added to associate y with the operands: $y \leq x_1 + x_2, y \geq x_1, y \geq x_2$. Given*

the general case of more operands $x_1, \dots, x_n$, it is possible to express the constraints as follows: $y \geq x_i \forall i \in \mathbb{N} : 1 \leq i \leq n, y \leq \sum_i^n x_i$.

Definition 4.2.10 (Logical NOT Linearization). Let us consider a boolean variable x . The logical NOT $\neg x$ can be expressed as $1 - x$.

Definition 4.2.11 (Logical XOR Linearization). Let us consider a logical XOR operation between two boolean variables operands x_1 and x_2 . To linearize the operation, the logical XOR $x_1 \oplus x_2$ shall be replaced using an auxiliary variable y , and the following constraints shall be added to associate y with the operands: $y \leq x_1 + x_2, y \geq x_1 - x_2, y \geq x_2 - x_1, y \leq 2 - x_1 - x_2$.

Definition 4.2.12 (Logical Implication Linearization). Let us consider a logical implication operation between two boolean variables operands x_1 and x_2 . To linearize the operation, the logical implication $x_1 \Rightarrow x_2$ can be simply rewritten as $x_1 \leq x_2$. Otherwise, a boolean variable y can be used such that $y = x_1 \rightarrow x_2 = \neg x_1 \vee x_2$, and the following linear constraints must be added: $y \leq 1 - x_1 + x_2, y \geq 1 - x_1, y \geq x_2$.

Definition 4.2.13 (Product of Real and Boolean Variables Linearization). Let us consider z a real decision variable such that $0 \leq z \leq U$, and x a decision variable. Then, the product zx can be replaced with a real variable y such that $y \leq Ux, y \leq z, y \geq z - (1 - x)U, y \geq 0$. If the upper bound U is unknown, then a big number can be used ('big- M ' approach).

Definition 4.2.14 (Maximum Linearization). Let us consider x_1 and x_2 two integer variables. Given m a discrete variable, the maximum of x_1 and x_2 can be expressed through the following linear constraints: $m \leq x_1 + x_2, x_1 \leq m, x_2 \leq m$. The general form, considering $x_1, \dots, x_n$ integer variables, can be expressed through the following linear constraints: $m \leq \sum_i^n x_i, x_i \leq m \forall i \in \mathbb{N} : 1 \leq i \leq n$.

4.2.7 Problem Formulation

Given the introduced concepts for CPS architectures tailored to automotive E/E architectures, which include definitions and equations, our problem statement for the optimization problem can be formalized as follows:

Definition 4.2.15 (MILP Problem Formulation). Let us define $\mathcal{R}_a$ the application requirements and $\mathcal{R}_b$ the basis requirements. Given a system template $\mathcal{T} = (V, E)$ and a library $\mathcal{L}$ of components, our problem consists in finding an assignment B^* (topology

configuration E^* , a map M^* and a routing R^*) such that all application constraints $\mathcal{R}_a$ and basis constraints $\mathcal{R}_b$ are satisfied.

Taking into account the linearization techniques as the final step in completing the mathematical formulation, we can now proceed to present the formal formulation of our optimization problem in the subsequent section.

4.3 Simulation Parameters and Metrics

In this section, we outline the key parameters used to evaluate the resulting architecture, detailing also the process of deriving simulation parameters from library data.

4.3.1 CAN Bus Load

We utilize simulations to measure the CAN bus load of the fixed part of our system network. Additionally, we perform analytical calculations to compare with the simulation data, ensuring consistency between the two methods. This is achieved using the following formula:

$$BL = \frac{1}{r} \sum_{n=1}^N \frac{L_n}{T_n} = \frac{1}{r} \sum_{n=1}^N F_n L_n \quad (4.6)$$

where r represents the bus baud rate, N is the number of CAN bus messages scheduled on the CAN bus, T_n is the transmission period of a message, L_n is the frame length and F_n is the number of frames per second. In other words, L_n/T_n (or equivalently $L_n F_n$) is the message bandwidth.

4.3.2 Camera Data Rate

In the optimization phase, the mapping matrix $\mathcal{M}$ associates the virtual components to real components specified in the library $\mathcal{L}$. For sensors, we have a range of devices including automotive radars and cameras. While the transmission data rate for radars is typically available in their manuals, most commercial camera manuals do not report the equivalent data rate. To correctly insert the equivalent data rate r of each camera in our component library $\mathcal{L}$, we calculate it as:

$$r = \frac{H_{px} V_{px} b_{depth} F_{ps}}{CR} \quad (4.7)$$

where H_{px} and V_{px} are the number of horizontal and vertical pixels (resolution), b_{depth} is the bit depth, F_{ps} represents the number of frame per second, and CR is the H264 compression ratio.

4.3.3 Latency and Jitter

To assess the performance of a network, it is crucial to understand the time it takes for a packet to travel from the source to the sink, known as latency. This, combined with the processing time required by the Central Processing Unit (CPU), helps determine if a task can meet its real-time deadline. This step is essential in designing functionalities for Advanced Driver Assistance Systems (ADAS) and autonomous driving features. Another important parameter is the jitter, which measures the variability in the time it takes for data to travel from one point to another. The jitter is critical for ensuring reliable communication. In our case, the latency of each stream is measured during simulation, while the jitter (calculated as the difference between successive latencies) is derived from equation (4.8):

$$J_k = |t_{sk_i} - t_{sk_{i-1}}|, i > 0 \quad (4.8)$$

4.3.4 Data Payload

After optimizing the architecture, the mapping matrix $\mathcal{M}$ indicates which library components from $\mathcal{L}$ are selected. However, in the simulation environment, we can only model the maximum frame size (payload) of a sensor, rather than the data rate directly. The data rate for each stream in the network for the Audio Video Bridging (AVB) protocol is defined in equation (4.9) [74]:

$$r = \frac{8MIF(MFS + PFO)}{CMI} \quad (4.9)$$

where CMI represents the Class Minimum Interval (125 μs for Class A and 250 μs for class B), MIF is the maximum interval frame period (number of frames sent in one CMI), MFS is the Maximum Frame Size and PFO is the Payload Frame Overhead (Header, Frame Check Sequence, etc.). To perform the simulation, we need to specify the MFS as a parameter. Therefore, after having calculated the equivalent rate from equation (4.7), we retrieve the MFS from equation (4.9).

For TTEthernet frames, since we know the payload of the control data and their scheduling period, we can directly specify these parameters in the simulation.

4.4 Constraints Reformulation

During the design phase, a crucial step is to determine the maximum bandwidth constraint for the backbone, which is the link between the central switch and the central processing unit (HPCU or ADCU), in order to obtain a feasible solution (all streams and messages can be scheduled and transmitted). As a first hypothesis, it is possible to set 75% as the maximum bandwidth limit for non-cross traffic specified by AVB standards. However, in the integrated system, the presence of TTEthernet frames and cross-traffic due to the conversion of CAN frames to Ethernet frames consistently leads to an unfeasible solution.

To find the maximum bandwidth constraint for which a feasible solution exists, the constraint can be modeled as a cost function, and an optimization algorithm can be used to locate it or approximate it. Choosing the right optimization algorithm is crucial, because otherwise it may perform no better than random search [107].

In our case, the bandwidth objective constraint is an increasing function, reaching a maximum and then dropping to zero when the solution becomes unfeasible. The cost function is therefore linear with an unknown slope and has a single maximum. Hence, it is not differentiable at that point. Several methods can be considered for maximizing the bandwidth constraint. However, each has its own strengths and limitations.

Brute force is not affordable due to the excessive simulation time required. Hill-climbing, which involves iterative steps to reach the maximum, may require numerous iterations and must be tuned differently depending on the location of the maximum, making it unsuitable for our problem. Newton's method, which leverages second-derivative information, can converge quickly to the maximum if the function is concave in its vicinity. However, since the considered cost function is linear, the second derivative is zero, making Newton's method unsuitable.

Stochastic methods, such as simulated annealing, genetic algorithms, and particle swarm optimization, are often effective for high-dimensional spaces with multiple peaks. They can escape local maxima and handle complex optimization problems. However, they involve random behavior that may slow down convergence. Given the high cost of simulations and the presence of only one maximum, stochastic approaches are not suitable for this problem.

Gradient-based methods can be used if the function is differentiable. These techniques iteratively adjust parameters to approach the maximum. Despite the unknown slope and non-differentiability at the peak, gradient ascent methods can still handle this issue, for example, by using an adaptive learning rate (step size).

The golden-section search, a deterministic technique for unimodal functions, optimizes without derivative information and requires relatively few iterations. While it is not the ideal choice due to requiring two evaluations of the objective function per iteration, it can still be used if necessary. However, given that we are dealing with a time-expensive simulation, this limitation must be carefully weighed.

Considering these factors, the gradient ascent method with an adaptive learning rate has been chosen. This approach allows the algorithm to dynamically adjust the step size, taking larger steps when far from the maximum and smaller steps as it approaches the peak. If the maximum is overshoot, the step size is reduced accordingly, and the process restarts from the previous temporary maximum point. Although this method may not converge exactly to the maximum, it can approximate it efficiently. In general, the gradient ascent method with an adaptive learning rate requires fewer simulations compared to the golden-section search, as will be demonstrated in the evaluation chapter.

To summarize, the gradient ascent method with an adaptive learning rate strikes a balance between convergence speed and computational cost. While it may not achieve the exact maximum, it offers a practical and efficient solution for our problem. In the next sections, the gradient ascent and golden-section search methods will be presented, while in 6, they will be compared based on the required number of simulations.

4.4.1 Adaptive Gradient Ascent Algorithm

As anticipated, we use a gradient ascent-based with adaptive learning rate method to reformulate the constraints and feed them back to the optimizer, to converge quickly near the maximum (Algorithm 1). The two main computational advantages of the gradient ascent algorithm are the simple implementation and the low storage requirements necessary. We now formalize the implementation of the algorithm, providing a detailed definition of the parameters of interest.

Let us assume we need to find the maximum of a function $f(x), x \in \mathbb{R}^n$, and $f : \mathbb{R}^n \rightarrow \mathbb{R}$. We denote the gradient of f by $g_n = g(x_n) = \nabla f(x_n)$. We compute a step (represented by n) along a given direction d_n :

$$x_{n+1} = x_n + \alpha_n d_n, \quad n = 0, 1, \dots, \quad (4.10)$$

For the gradient ascent method, the search direction is given by:

$$d_n = g_n = \nabla f(x_n) \quad (4.11)$$

4.4. Constraints Reformulation

The step length parameter α_n , also known as the learning rate, is adapted at each iteration as follows:

$$\alpha_n = \begin{cases} \alpha_{n-1} & \nabla f(x_n) > \epsilon \\ \frac{\alpha_{n-1}}{K} & \nabla f(x_n) < -\epsilon \end{cases} \quad (4.12)$$

where ϵ is the convergence tolerance. Therefore, convergence is reached when:

$$|\nabla f(x_n)| \leq \epsilon. \quad (4.13)$$

During the evaluation, we will demonstrate how using this algorithm can lead to fewer design iterations and, consequently, reduced design time. This effect is particularly evident in larger scenarios where a significant number of sensors are added to the network.

Algorithm 1 Gradient Ascent Algorithm with Adaptive Learning Rate

```

 $\alpha_n \leftarrow \alpha_0$ 
while  $g_n \leq \epsilon$  and  $iter < N$  do
    Evaluate cost function  $x_n \leftarrow f(x_n)$ 
    Calculate gradient  $g_n \leftarrow \nabla f(x_n) = x_n - x_{n-1}$ 
    if  $x_n \neq 0$  then ▷ Solution Feasible
         $x_{n-1} \leftarrow x_n$ 
    else ▷ Solution Unfeasible
         $\alpha_n \leftarrow \frac{\alpha_n}{K}$ 
         $x_n \leftarrow x_{n-1}$ 
    end if
    Update  $x_n \leftarrow x_n + \alpha_n g_n$ 
end while

```

4.4.2 Golden-Section Search

The Golden-Section Search is a mathematical optimization algorithm which is used to find the maximum or minimum of an unimodal function (a function with only one peak or one valley) over a specified interval (Algorithm 2). It is a derivative-free method, meaning that it does not require the computation of derivatives. Therefore, it is ideal for optimizing non-differentiable or noisy functions.

The algorithm is based on the golden ratio, $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$, which divides an interval into two parts so that the ratio of the larger part to the smaller part is the same as the ratio of the whole interval to the larger part. This property ensures reduction of the search space at each step in an efficient way.

Algorithm 2 Golden-Section Search Algorithm

```

 $\phi \leftarrow \frac{1+\sqrt{5}}{2}$  ▷ Golden ratio constant
Initialize  $a \leftarrow a_{\min}$ ,  $b \leftarrow b_{\max}$  and tolerance  $\epsilon$ 
while  $|b - a| \geq \epsilon$  do
     $it \leftarrow it + 1$ 
    Compute intermediate points:
     $x_1 \leftarrow a + (1 - 1/\phi) \cdot (b - a)$ 
     $x_2 \leftarrow a + (1/\phi) \cdot (b - a)$ 
    Evaluate cost function:
     $f^*(x_1) \leftarrow f(x_1)$ 
     $f^*(x_2) \leftarrow f(x_2)$ 
    if  $f^*(x_1) > f^*(x_2)$  then
        Update  $b \leftarrow x_2$ 
    else
        Update  $a \leftarrow x_1$ 
    end if
end while
Compute maximum:  $maximum \leftarrow (a + b)/2$ 

```

A key drawback is the need to compute the objective function twice per iteration, which is a significant challenge in our case due to the time-consuming nature of the simulations.

4.5 Trade-Off Evaluation

To evaluate trade-offs between power consumption, cost, and weight, multi-objective optimization methods are often applied. These methods produce a Pareto front, representing a set of optimal solutions where no single solution is superior to another unless one objective improves at the expense of another. For example, minimizing power consumption might lead to increased cost or weight, highlighting the inherent trade-offs between objectives. As it will be demonstrated in Chapter 6, we will prioritize minimizing power consumption, while cost and weight constraints will be varied to explore different solutions, ultimately constructing a Pareto front. Several methods can address this. In small design spaces with limited decision variables, a brute force approach using a sufficiently small step size may yield robust Pareto surfaces. However, this method lacks guarantees against overlooking better, non-simulated points, as they may lie between sampled steps. Employing an even smaller step size might resolve this issue but significantly increases computational demand, particularly for resource-intensive sim-

ulations. To enhance the tool scalability and to improve exploration capabilities in larger design spaces, different approaches can be adopted. In the following subsections, we will formally describe two multi-objective optimization methods: Non-dominated Sorting Genetic Algorithm II (NSGA-II) and Multi-objective Bayesian Optimization (MOBO). Subsequently, in Chapter 6, these methods will be compared against the brute-force approach.

4.5.1 Non-Dominated Sorting Genetic Algorithm II (NSGA-II)

Introduced by Deb et al. in 2002, the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [18] is a robust evolutionary methodology designed to tackle multi-objective optimization problems. It is particularly efficient in addressing trade-offs between conflicting objectives, providing a well-distributed set of Pareto-optimal solutions.

In general, evolutionary algorithms draw inspiration from the principles of natural selection and genetics observed in biological evolution. The process of the NSGA-II begins with initialization, where an initial population of solutions is randomly generated within the specified constraints. Following this, the non-dominated sorting step ranks the solutions based on their dominance relationships. To ensure diversity among the solutions, the crowding distance calculation measures the proximity of each solution to its neighbors. This step helps in maintaining a diverse set of solutions. Subsequently, the steps of selection, crossover, and mutation are executed to generate new solutions. Selection identifies the best solutions, while crossover combines parts of two parent solutions to create new offspring. Mutation introduces random changes to some solutions to maintain diversity. The concept of elitism ensures that the best solutions are preserved from one generation to the next. Throughout this process, solutions are sorted into different fronts based on their dominance, ensuring that the most optimal solutions are identified.

Before delving into the details of the algorithm, it is essential to introduce some key definitions.

Definition 4.5.1 (Non-Dominated Sorting). *A solution x_i dominates x_j if $\forall k \in \{1, \dots, M\}, f_k(x_i) \leq f_k(x_j)$ and $\exists k \in \{1, \dots, M\}, f_k(x_i) < f_k(x_j)$, where f_k is the k -th objective function.*

Definition 4.5.2 (Crowding Distance). *For each solution x_i , the crowding distance d_i is calculated as $d_i = \sum_{k=1}^M \left(\frac{f_k^{i+1} - f_k^{i-1}}{f_k^{max} - f_k^{min}} \right)$, where f_k^{i+1} and f_k^{i-1} are the objective values of the neighboring solutions in the sorted list for the k -th objective.*

Definition 4.5.3 (Crossover). For each pair of parent solutions $\mathbf{x}_1$ and $\mathbf{x}_2$, the crossover operation is performed with probability α . A crossover point p is chosen randomly within the range $[1, n - 1]$. The child solutions $\mathbf{c}_1$ and $\mathbf{c}_2$ are generated as follows:

$$\begin{aligned}\mathbf{c}_1 &= \mathbf{x}_1[1 : p] \cup \mathbf{x}_2[p : n], \\ \mathbf{c}_2 &= \mathbf{x}_2[1 : p] \cup \mathbf{x}_1[p : n],\end{aligned}$$

where $\mathbf{x}_1$ and $\mathbf{x}_2$ are the selected parent solutions.

Definition 4.5.4 (Mutation). For each gene i in an individual solution $\mathbf{x}$, the mutation operation is performed with probability β . If $r < \beta$, the new gene value x_i is generated as $\text{uniform}(l_i, u_i)$, where r is a random number uniformly distributed between 0 and 1, and l_i and u_i are the lower and upper bounds for the gene i .

Considering the definitions introduced earlier, the primary steps of the algorithm are formally summarized in Algorithm 3.

Algorithm 3 Non-dominated Sorting Genetic Algorithm II (NSGA-II)

Initialize population P_0 of size N

Evaluate fitness $f(\mathbf{x}_i)$ for each individual $\mathbf{x}_i$ in P_0

for each generation **do**

 Perform non-dominated sorting

 Calculate crowding distance d_i :

$$d_i = \sum_{k=1}^M \left(\frac{f_k^{i+1} - f_k^{i-1}}{f_k^{\max} - f_k^{\min}} \right)$$

 Select parents based on crowding distance and non-domination rank

for each pair of parents $\mathbf{x}_1, \mathbf{x}_2$ **do**

 Perform crossover with probability α

$$\mathbf{c}_1 = \mathbf{x}_1[1 : p] \cup \mathbf{x}_2[p : n], \quad \mathbf{c}_2 = \mathbf{x}_2[1 : p] \cup \mathbf{x}_1[p : n]$$

 Perform mutation with probability β

$$x_i = \text{uniform}(l_i, u_i) \text{ if } r < \beta$$

end for

 Combine populations: $R_t = P_t \cup Q_t$ (Elitism)

 Evaluate fitness $f(\mathbf{x}_i)$ for each individual $\mathbf{x}_i$ in R_t

 Sort R_t based on non-domination

 Select the top N individuals to form the new population P_{t+1}

end for

4.5.2 Multi-objective Bayesian optimization (MOBO)

Multi-objective Bayesian optimization (MOBO) is an approach designed to find the best balance among several conflicting objectives, particularly when their evaluation is resource-intensive. Also in this case the objective is to determine the Pareto front. In general, MOBO uses surrogate models, like Gaussian Processes, to approximate the objective functions and acquisition functions to guide the search for optimal solutions. Before summarizing a basic implementation for the algorithm, some key concepts must be introduced.

In the method adopted in this work, the Gaussian Process (GP) is used as a surrogate model to approximate the objective functions. It is defined as:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \quad (4.14)$$

where $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ is the mean function and $k(\mathbf{x}, \mathbf{x}') = \text{Cov}(f(\mathbf{x}), f(\mathbf{x}'))$ is the covariance function (kernel).

Given the training dataset $\mathcal{D} = \{(\mathbf{X}, \mathbf{y})\}$, where $\mathbf{X}$ are the decision variables (inputs) and $\mathbf{y}$ are the evaluated objective functions (outputs), the covariance matrix is computed as:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & \cdots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix}, \quad (4.15)$$

with added noise:

$$\mathbf{K}_{\text{train}} = \mathbf{K} + \sigma^2 \mathbf{I}, \quad (4.16)$$

where σ^2 is the noise variance and $\mathbf{I}$ is the identity matrix.

The covariance function $k(\mathbf{x}, \mathbf{x}')$ used in the Gaussian Process combines a constant kernel and a radial basis function (RBF) kernel:

$$k(\mathbf{x}, \mathbf{x}') = C \cdot \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right), \quad (4.17)$$

where C is the scaling factor, with $C \in [C_{\min}, C_{\max}]$, ℓ is the length scale, with $\ell \in [\ell_{\min}, \ell_{\max}]$, and $\|\mathbf{x} - \mathbf{x}'\|$ is the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}'$.

For a new candidate point $\mathbf{x}^*$, the predictions are given by the predictive mean and variance. The predictive mean is defined as:

$$\mu(\mathbf{x}^*) = \mathbf{k}^\top \mathbf{K}_{\text{train}}^{-1} \mathbf{y}, \quad (4.18)$$

where $\mathbf{k}$ is the covariance vector between $\mathbf{x}^*$ and the training points.

The predictive variance is defined as:

$$\sigma^2(\mathbf{x}^*) = k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}^\top \mathbf{K}_{\text{train}}^{-1} \mathbf{k}. \quad (4.19)$$

At the end of the iterations, the Pareto-efficient solutions $\mathbf{X}_{\text{Pareto}}$ and $\mathbf{Y}_{\text{Pareto}}$ are identified, ensuring no objective can be improved without degrading another.

The basic implementation for MOBO is shown in Algorithm 4.

Algorithm 4 Multi-objective Bayesian Optimization (MOBO)

Input: Parameter bounds $\mathbf{L}$ (lower) and $\mathbf{U}$ (upper), number of iterations N_{iter}
Initialize random samples $\mathbf{X}_0 = \{\mathbf{x}_i\}_{i=1}^{N_{\text{init}}}$ from the parameter space $[\mathbf{L}, \mathbf{U}]$
Evaluate objectives $\mathbf{Y}_0 = \{f(\mathbf{x}_i)\}_{i=1}^{N_{\text{init}}}$ for all initial samples
Train: Fit Gaussian Process (GP) to surrogate model $\mathbf{Y} = g(\mathbf{X})$
for each iteration $t = 1, \dots, N_{\text{iter}}$ **do**
 Optimize Acquisition Function: Solve $\mathbf{x}^* = \arg \min_{\mathbf{x} \in [\mathbf{L}, \mathbf{U}]} a(\mathbf{x}; g)$
 Evaluate Objectives: $\mathbf{y}^* = f(\mathbf{x}^*)$
 Add $\mathbf{x}^*$ and $\mathbf{y}^*$ to the dataset: $\mathbf{X} \leftarrow \mathbf{X} \cup \{\mathbf{x}^*\}$, $\mathbf{Y} \leftarrow \mathbf{Y} \cup \{\mathbf{y}^*\}$
 Retrain GP with updated $\mathbf{X}$ and $\mathbf{Y}$
end for
Output: Pareto-efficient solutions $\mathbf{X}_{\text{Pareto}}$, $\mathbf{Y}_{\text{Pareto}}$ using Pareto dominance

Chapter 5

Design Space Exploration Framework

In this chapter, we provide a comprehensive overview of the design space exploration framework that we developed, focusing on three key components: the automated design toolbox, the simulation framework, and the interface methodology between the two environments. We will focus on describing the software components implemented, the libraries used within the simulation environment, and the methodology used to implement communication between the optimizer and the simulator.

5.1 Conceptual Structure of the Framework

A conceptual scheme of the design space exploration framework is shown in Figure 5.2.

As anticipated in the introduction of this dissertation, we have developed a design space exploration framework based on the ArchEx tool [63, 43], extending and adapting it to meet the project requirements. ArchEx, originally a MATLAB toolbox utilizing YALMIP¹ as the solver interface, faced compatibility issues with recent MATLAB versions. Consequently, we reimplemented the framework in Python, incorporating the essential constraints for our project and introducing new ones as needed. In place of YALMIP, we opted for the CVXPY² Python library to handle MILP modeling and solving, using the CPLEX optimizer.

In particular, we further adapted and extended the framework to support the verification through simulation of zone-based E/E topologies for Off-Road Vehicles (ORVs), Drive-by-Wire (DBW) guidance systems, and heterogeneous networks incorporating CAN, AVB, and TTEthernet protocols.

¹<https://yalmip.github.io/>

²<https://www.cvxpy.org/>

5.1. Conceptual Structure of the Framework

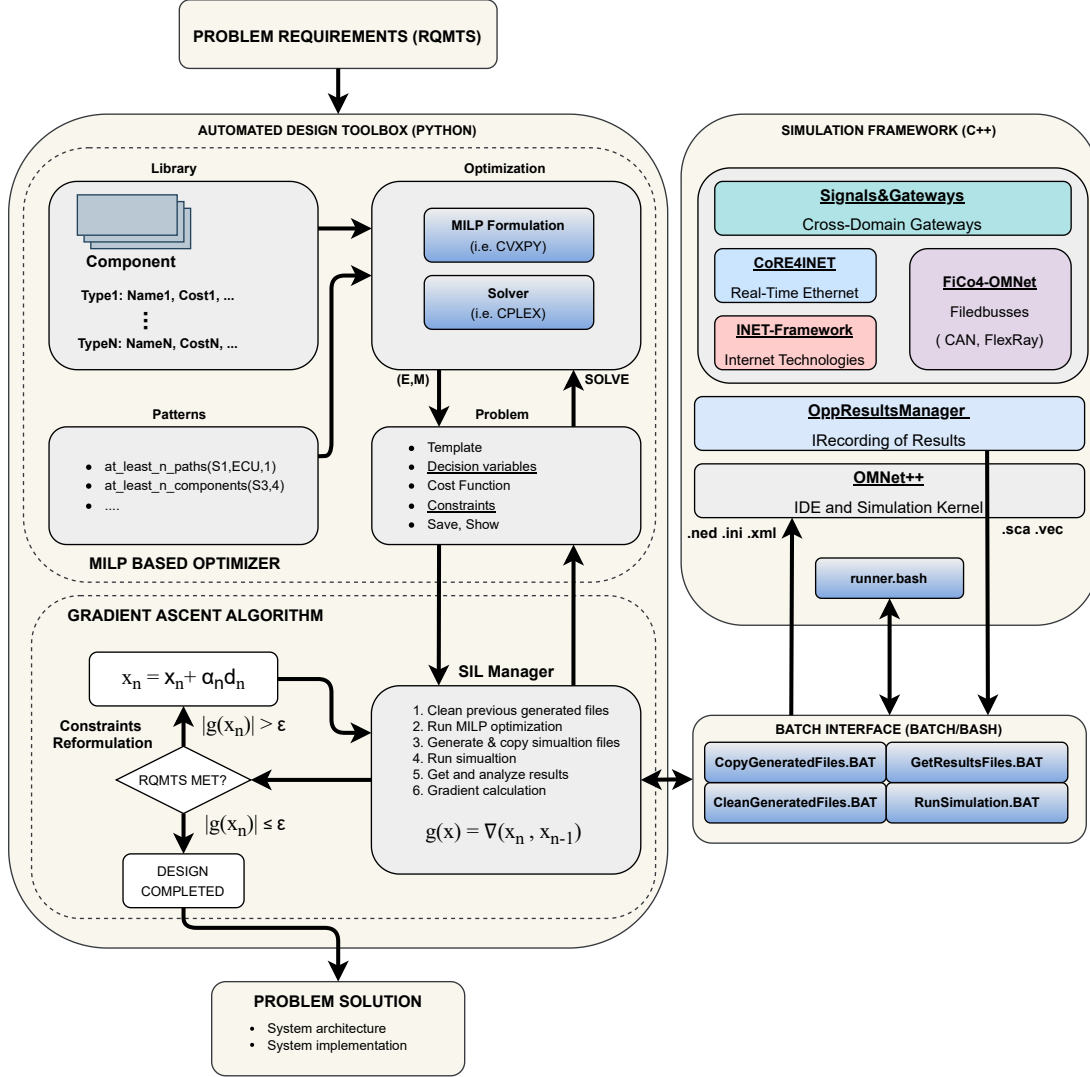


Figure 5.1: Toolbox Conceptual Structure

The framework consists of three main parts:

- **Automated Design Toolbox.** Implemented in Python, this component of the framework serves as the methodology coordinator. It receives design specifications as input, processes them, and models them for the optimizer. The optimizer outputs, which includes both the architecture and implementation details, are then used to generate configuration files for the verification process, which is conducted via simulation.

At the end of the simulation, the resulting data are collected and analyzed. If a satisfactory solution is found, it is presented to the user. Otherwise, a constraint

reformulation procedure is initiated using a gradient-based search strategy. This iterative process continues until convergence to an optimal solution is achieved. Once the input problem files are defined, the entire workflow-from optimization to simulation and iterative design exploration-is fully automated.

- **Simulation Framework.** Implemented in C++, the CoRE-RG framework³ enables the simulation of heterogeneous vehicular networks, including CAN, AVB, and TTEthernet communication standards. Using configuration files, we define the system topology and component specifications, as determined by the optimizer. This allows us to verify whether parameters such as latency, jitter, and network load meet the required criteria, as well as monitoring other parameters of interest.
- **Batch Interface.** In Windows, the CoRE-RG framework must be executed via a MinGW terminal, preventing direct simulation runs from Python code. To bridge this gap, we developed Batch scripts that serve as an interface between the Python code and the simulation environment. These scripts facilitate the initiation and monitoring of simulations, execution of subroutines, and retrieval of simulation data. Additionally, they enhance the framework flexibility by decoupling the two domains, thereby simplifying future extensions.

As a result, the framework enhances the design automation related to off-road vehicle E/E architectural design. In fact, the user specifies project requirements (design constraints like cost, weight, power consumption, etc.), template structure (connection rules among components) and components library (list of COTS components) through two textual files, enabling the optimizer to generate candidate solutions through various patterns. Simulation are automatically managed and results are subsequently automatically retrieved and analyzed. The final design solution is determined by iteratively leveraging these two methods (i.e. optimization and simulation) in a feedback loop. In the following sections, we provide implementation details along with comprehensive examples to offer a broad understanding of the DSE framework, including software components description, library objects of the simulation environment and design choices.

³<https://github.com/CoRE-RG>

5.2 Automated Design Toolbox Architecture

A conceptual scheme of the automated design toolbox software architecture is shown in Figure 5.2. The structure of the software is designed to be flexible and application independent, so that through the use of generic classes, it is possible to handle various mapping problems.

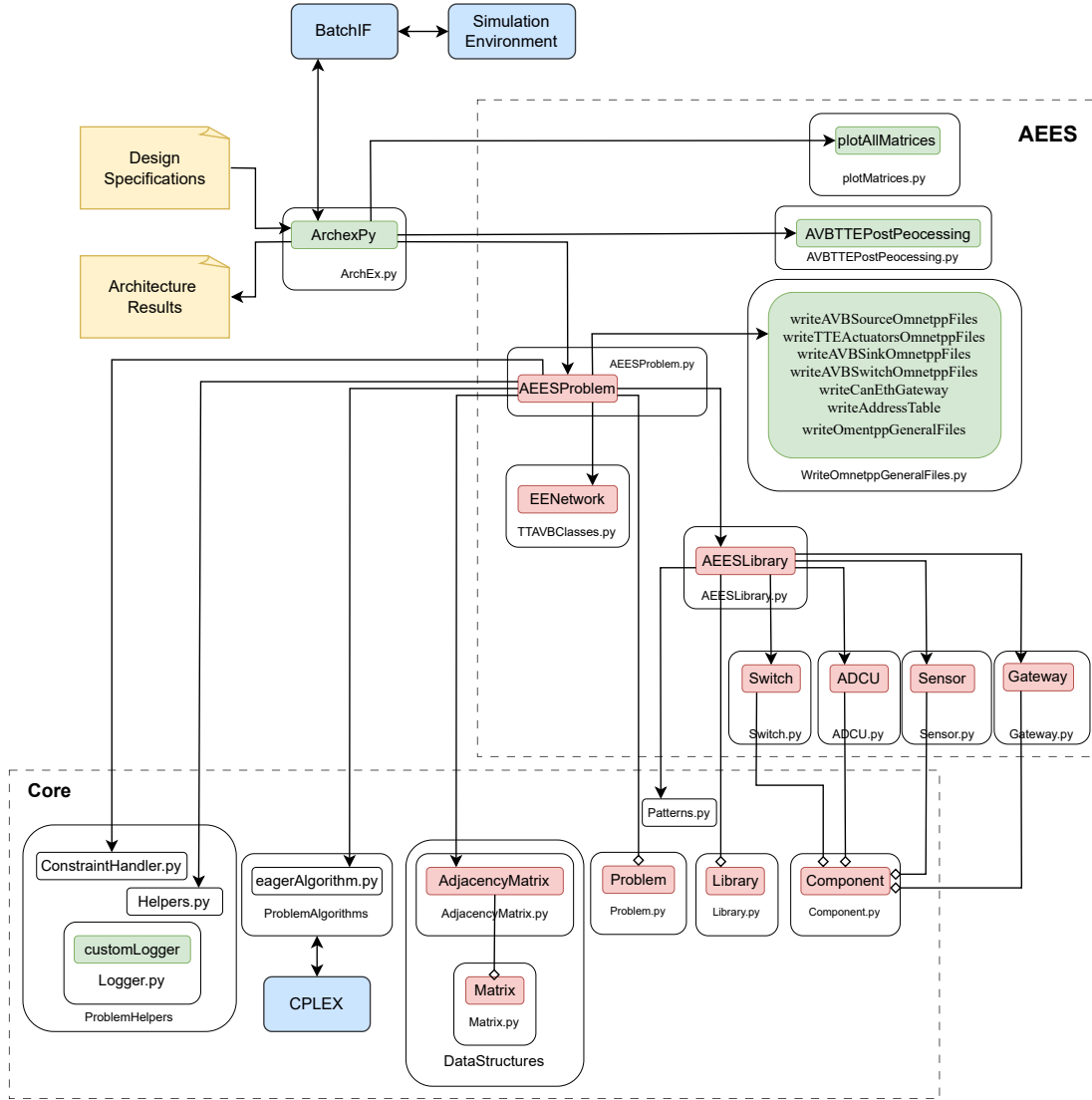


Figure 5.2: Conceptual Software Architecture

The SW input (design specifications) consists of two text files (library and problem description), in which the user specifies general information (types, functional flows, tags), the structure of the model (for example, connection rules), and the requirements

of the problem through the patterns (i.e., power consumption, cost, weight, and rate constraints). The library is arranged to work with several component types, in which each record represents a distinct real component, including both a name and a certain number of attributes. Through tagging, components of the same type can be grouped for domain-specific problems.

In our project, we consider a zone-based E/E automotive topology, thus we distinguish between front-center, center-right, center-left and rear-center sensors, and Ethernet switches, based on the vehicle location, placing restrictions on feasible connections. Both types and tags can be used as parameters in the template formulation. The Software (SW) output (architecture results) are instead the resulting system topology and the mapping between virtual and real components. In the next section we provide a detailed description of the software components.

5.2.1 Software Components

In this section, we provide details of each software component depicted in Figure 5.2. To improve the readability of the dissertation, we avoid fully providing the implemented python code. However, we report when needed some code listings which could help the reader to understand how the implementation was fulfilled.

- **Component.** The class *Component* serves as a foundational structure, encompassing essential attributes such as name, cost, and subtype, along with methods for managing a network component. Derived from this class are the subclasses Sensor, Switch, Gateway, and ADCU, each of which extends the base class by incorporating specific attributes like weight, power consumption, and data rate.
- **Library.** The *Library* class serves as a base model for constructing a library object. It utilizes Python dictionaries to store various attributes, including the list and count of components, their types, subtypes, names, weights, power consumption, and data rates.
- **Patterns.** The *Patterns* module provides functions to transform constraints into numerical representations, thereby converting them into linear constraints suitable for inclusion in mixed-integer programming problems.

The constraints are written in text format. Therefore, the associated text strings shall be first parsed and then be fed to the pattern function prototypes which return the MILP of constraints.

For example, constraint 4.5a, which is used to ensure that each instantiated virtual components in the system architecture have a mapping (i.e. they are associated to a real COTS component in the library), can be implemented in the SW like follows:

```
import numpy as np
import cvxpy as cp

# - condition is a trasnpose slice (subset of a particular
#   type, like Sensor) of the adjacency matrix
# - target is the mapping matrix
def if_used_then_has_mapping( condition, target ):
    cons = []
    # Maximum linearization
    for i in range(0, condition.shape[1]):
        x1 = condition[:,i]
        y1 = target[:,i]

        m1 = cp.Variable()
        m2 = cp.Variable()

        cons.append( m1 <= np.sum(x1) )
        cons.append( m2 <= np.sum(y1) )

        for i in range(0, len(x1)):
            cons.append(x1[i] <= m1)

        for i in range(0, len(y1)):
            cons.append(y1[i] <= m2)

        cons.append(m1 == m2)

    milp_constraint = cons

    return milp_constraint
```

Listing 5.1: Code Example of Constraint 4.5a SW Implementation

Note that, the linearization of the maximum function, explained in definition 4.2.14, is used to implement this particular constraint.

Since the maximum functionality is not provided in the linear formulation that we need within CVXPY, we implemented it directly inside the pattern code.

In addition, constraint 4.5b, which is used to ensure that each virtual component is implemented with no more than one real component, can be implemented in the software like follows:

```
import numpy as np

def atMost( op1, op2 ):
    """
    OP1 is smaller than or equal to OP2
    Function atMost takes two parameters OP1 and OP2 and returns
    the CVXPY inequality OP1 <= OP2.
    """
    milp_constraints = []

    if (len(op1) != len(op2)):
        print('[error]: Len op1 != len op2')

    for i in range(0, len(op1)):
        milp_constraints.append( op1[i] <= op2[i] )

    return milp_constraints

def at_most_one_mapping( M ):
    s = M.shape[1]
    const = np.ones(s, dtype='int')

    milp_constraint = atMost(np.sum(M, axis=0), const)

    return milp_constraint
```

Listing 5.2: Code Example of Constraint 4.5b SW Implementation

- **AEESLibrary.** The class *AEESLibrary* extends the *Library* class to represent the problem library $\mathcal{L}$. It includes instances of the *Sensor*, *Switch*, *Gateway*, and *ADCU* classes, and provides methods to interact with these components. Additionally, this module includes all the necessary customizations to model an Automotive Electrical and Electronic System (AEES) component library. It implements methods to parse and store the library specification file, create mapping matrices between 'virtual components' (columns) and 'real components' (rows), and add the encoded constraints 4.5a and 4.5b starting from the mapping matrices.

- **ConstraintHandler.** The *ConstraintHandler* module includes methods for parsing constraints from the input problem file and converting them into a unified format structure. It returns MILP formulations which are input for the *Patterns* and classifies the constraints into corresponding groups (such as Flow, Interconnection, etc.).
- **Matrix.** The *Matrix* class serves as a general object for representing and handling matrices. It provides methods for managing the matrix, where each element is represented by binary decision variables. Using Python dictionaries, it is possible to fetch rows and columns, retrieve specific elements of the matrix, and obtain the matrix numerical form. This representation can be useful for plotting the final architecture and identifying which connections or components are instantiated.
- **AdjacencyMatrix.** The *AdjacencyMatrix* class, inherited from the class *matrix*, is used to represent the adjacency matrix E of the problem, which model the system architecture. Each element in this matrix represents the connectivity between network components, modeled using binary decision variables. Following the MILP optimization, these boolean variables, which initially can take on values of either 0 or 1, will assume a value of 1 if the connection between two components is established, and 0 if it is not. For example, a matrix of binary variables can be initialized as follows:

```
import numpy as np
import cvxpy as cp

# return a 2D Matrix of CVXPY binary variables
def createMatrixifOfBinaryVariables( row, cols):
    num_type1 = row
    num_type2 = cols
    bvar = []
    for i in range(0,num_type1):
        row_var = []
        for j in range(0,num_type2):
            var = cp.Variable(boolean=True)
            row_var.append(var)
        bvar.append(np.array(row_var))
    bvar = np.array(bvar)

    return bvar
```

Listing 5.3: Code Example of a Matrix made of Binary Variables

- **eagerAlgorithm.** The *eagerAlgorithm* module implements a monolithic optimization routine, which composes the problem using all provided constraints and tries to solve it in one shot ("eagerly"). It allows to call the MILP solver (i.e. CPLEX) via the CVXPY interface. The input of the solver are the cost function and the problem constraints (which are composed of both binary variables and integer linear constraints), while the output are an assignment of 0 or 1 to these variables.

```
import cvxpy as cp

def eagerAlgorithm( problem, options ):

    # 1. Define Objective through CVPY Interface
    objective = cp.Minimize(problem.CostFunction)
    # 2. Define Constraints
    constraints = problem.Constraints
    # 3. Create problem Instance
    prob = cp.Problem(objective, constraints)
    # 4. Solve the problem
    result = prob.solve(solver='CPLEX')

    if str(result) == 'inf':
        print('Solution Unfeasible')
        return
    else:
        print('A solution exists')
```

Listing 5.4: Code Example of the eagerAlgorithm

- **Helpers.** The *Helpers* module includes helper functions designed to manage various tasks. For instance, it implements routines to extract matrix slices of specific component types, facilitating the application of mapping constraints.
- **Problem.** The *Problem* class provides several routines to formulate and solve architectural exploration problems, such as parsing input files, specifying the component library, creating decision variables, formulating constraints and cost functions, and visualizing result data.
- **AEESProblem.** The *AEESProblem* class, inherited from *Problem*, extends the software capabilities to manage the specific application of E/E architecture design. During the object creation, it manage MILP problem formulation. During object

creation, it manages the formulation of the MILP problem. This includes processing the problem description and template, creating decision variables, handling problem constraints, formulating the cost function, and solving the MILP problem. Additionally, from the optimization results, it creates objects to represent the resulting architecture, which are then used to generate simulation files.

- **EENetwork.** The *EENetwork* class, instantiated in the *AEESProblem* class, is used to model and store the optimized architecture with software structures that are easily manageable. This serves as a foundation for the automatic generation of simulation file code.
- **WriteOmnetppGeneralFiles.** The *WriteOmnetppGeneralFiles* module contains a set of functions which starting from the *EENetwork* object write all the required files to run the simulation (.ini, .ned and .xml).
- **AVBTTEPostProcessing.** The *AVBTTEPostProcessing* module contains all the post-processing routines necessary for analyzing simulated data. It includes parsing .csv files related to simulation results, checking the feasibility of the solution by verifying if all streams are transmitted, and returning appropriate software return code if the solution is not feasible. It extracts key parameters such as the latency of simulated streams, network load, and calculates the jitter for each stream. Additionally, it plots the results data to assist the designer in analyzing the outcomes.
- **ArchExPy.** The *ArchExPy* module is the core part of the code. It contains the path of the input files and instantiates the *AEESProblem* object, which manages the entire optimization problem. Additionally, after the optimization phase, it handles the invocation of routines to generate the code for simulation (implemented in the *WriteOmnetppGeneralFiles* module) and the post-processing code to analyze the resulting data (implemented in the *AVBTTEPostProcessing* module). It also uses the batch interfaces to run and monitor the simulation. Finally, it implements the gradient-based search method to accelerate the design exploration process.

To complement the toolbox, additional modules are included to simplify software debugging. These modules include *plotMatrices* and *customLogger*.

5.3 Simulation Framework

This section provides an in-depth explanation of the simulation framework, including illustrative examples, the necessary configuration files, and instructions for executing simulations via the MinGW terminal.

As discussed in previous chapters, we selected the CoRE-RG framework to perform the simulations and validate the results obtained by the optimizer. This framework is built on OMNeT++, an extensible, modular, component-based C++ simulation library and framework primarily used for constructing network simulators.

OMNeT++ supports the development of project extensions as independent modules, enabling the modeling and simulation of a wide range of projects, from Wireless Sensor Networks to real-time communications. The INET extension includes models for the Internet stack, encompassing wired protocols such as Ethernet. In Figure 5.3 a conceptual diagram of the simulation environment is shown.

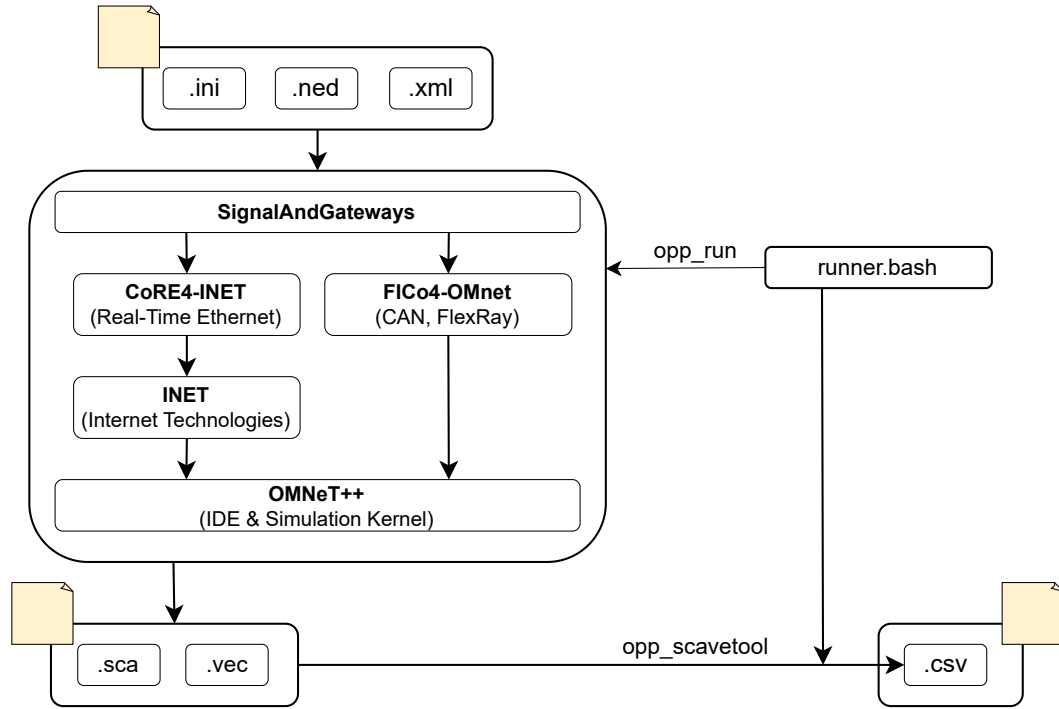


Figure 5.3: Simulation Environment

The CoRE-RG group has developed an extension for INET to model real-time Ethernet communications, including TTEthernet (AS6802) and AVB/TSN. Additionally, the CoRE-RG group has created an extension for OMNeT++ to model fieldbus communi-

cation standards, specifically CAN and FlexRay. To facilitate communication between these two domains (real-time Ethernet and CAN), they developed another extension called SignalAndGateways. This extension enables the modeling of inter-domain gateways, allowing the encapsulation of CAN payloads into Ethernet packets and vice versa.

In general, the framework utilizes two main file types for simulations. The .ned files specify the network topology and the hardware configuration of each node within the network. Conversely, the .ini files are used to define the parameters of the components or the network itself. Additionally, to model certain components, such as the inter-domain gateway, an .xml file is specified for its design specifications.

The framework provides an Eclipse-based Graphical User Interface (GUI) to facilitate the configuration and simulation of the network. Additionally, it offers methods to run simulations without the GUI, thereby reducing computational resource usage and speeding up simulation time. The primary result files generated are .sca files, which contain scalar values, and .vec files, which contain vector data. There are routines available to convert these formats into .csv files, making them more manageable for further processing with external software such as Python.

In the next sections, we provide examples of configuration files for each network component.

5.3.1 TTAVBEthernetHost

We use the *TTAVBEthernetHost* component to model both sensors, actuators and the processing ECU, which transmit or receive real-time Ethernet traffic. A schematic of the hardware model is shown in Figure 5.4

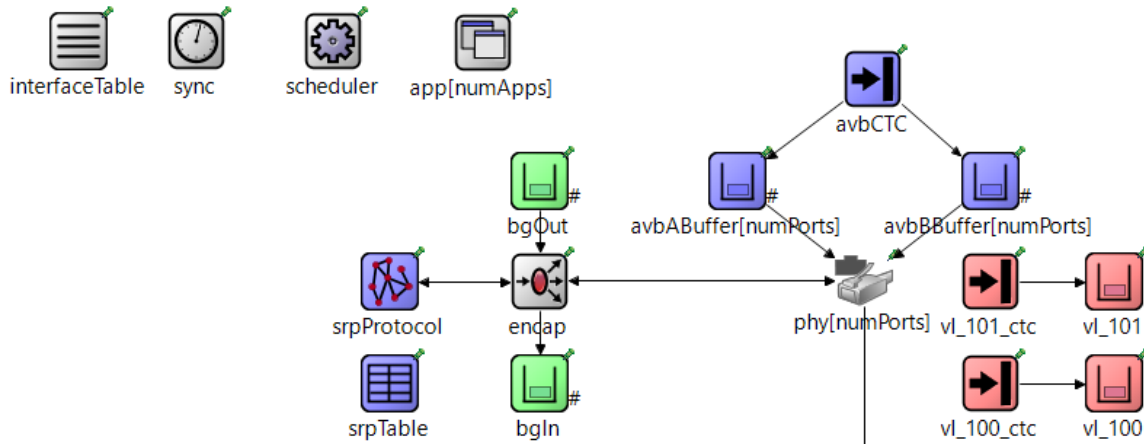


Figure 5.4: TTAVBEthernet Host .ned Model (GUI)

The component HW model defined through the .ned file include the following main modules:

- **Physical Port (phy)**: This is the physical port for the real-time Ethernet device.
- **AVB Queue Buffers (avbABuffer, avbBBuffer)**: The buffers handle Class A and Class B Audio Video Bridging (AVB), respectively. These buffers use the Credit Based Shaper algorithm to manage message transmission.
- **Conformance Check Module (avbCTC)**: This module performs conformance checks on incoming audio-video bridging traffic and routes AVB frames to the appropriate buffers.
- **SRP Protocol (srpProtocol)**: Handles the SRP protocol.
- **Background Traffic Queue Buffers (bgOut, bgIn)**: Input/Output First In First Out (FIFO) queues where messages are forwarded immediately.
- **Encapsulation Module (encap)**: This module forwards frames to the appropriate ports based on their destination MAC addresses. Incoming SRP frames are forwarded as SRP messages through the srpOut gate to the SRProtocol module. Outgoing SRP messages received through the srpIn gate are forwarded as Ethernet frames according to the control information.
- **TT Conformance Checkers (vl_100_ctc, vl_101_ctc)**: Modules for the conformance checks of incoming TT (Time-Triggered) traffic. Note that a network component can include one or more of these modules, depending on the specific application requirements.
- **TT Double Buffers (vl_100, vl_101)**: Represents a double buffer for time-triggered (TT) traffic, where messages are transmitted within the appropriately configured time windows. Note that a network component can include one or more of these modules, depending on the specific application requirements.
- **TT Scheduler (scheduler)**: TTEthernet scheduler module interface, which allows to manage time-triggered traffic.
- **Synchronization Module (sync)**: A synchronization module for the scheduler.
- **Application Manager (app)**: Application model to implement one or more applications (which can send or receive AVB or TT traffic).

- **Interface Table (InterfaceTable):** Maintain the table of IPV4 and IPV6 network interfaces.

A general structure of the code for a .ned file can be specified as follows:

```
package signalsandgateways.examples.SIMTTAVB0;

//used modules inclusion
import core4inet.nodes.ethernet.avb_AS6802.TTEAVBEtherHost;
import core4inet.incoming.AS6802.TTIncoming;
import core4inet.buffer.AS6802.TTDoubleBuffer;

//configuration in ecu1.ini.
module Ecu1 extends TTEAVBEtherHost
{
    // define incoming/outcoming buffers
    submodules:
        vl_100_ctc: TTIncoming {
            parameters:
        }
        vl_100: TTDoubleBuffer {
            parameters:
        }
        vl_101_ctc: TTIncoming {
            parameters:
        }
        vl_101: TTDoubleBuffer {
            parameters:
        }
    // define connections
    connections:
        vl_100_ctc.out --> vl_100.in;
        vl_101_ctc.out --> vl_101.in;
}
```

Listing 5.5: Code Example for HW Configuration

In the .ned file, which specifies mainly the HW configuration of the node, the main required packages are imported. Then, the number and configuration of the incoming/outcoming buffers is specified. In this example, two incoming/outcoming buffer are used. The other configuration must not be specified because the module is derived from the TTEAVBEtherHost.

In the other nodes, the same file is used with the opportune modifications or adaptations.

Regarding the .ini file, an AVB sensor sending AVB data (which can be for a Radar or a Camera) can be configured as follows:

```
# Network name
network = SIMTTAVB0

# MAC address definition
**.SensorMiddleRight0.phy[*].mac.address = "0E-03-00-00-00-00"
# Defines the number of running real-time applications.
**.SensorMiddleRight0.numApps = 1

#App: Talker (Send AVB Data of class A)
**.SensorMiddleRight0.app[0].typename = "AVBTrafficSourceApp"
**.SensorMiddleRight0.app[0].streamID = 0
**.SensorMiddleRight0.app[0].srClass = "A"
**.SensorMiddleRight0.app[0].intervalFrames = 1
**.SensorMiddleRight0.app[0].payload = 64Byte
```

Listing 5.6: Code Example for Sensor HW Initialization

A central processing ECU (like an HPCU or ADCU) which receives two AVB streams and sends TT control data to an actuator, can be specified for example through an .ini file looks as follows:

```
# Network name
network = SIMTTAVB0

# MAC address definition
**.ecu1.phy[*].mac.address = "0D-00-00-00-00-01"
# Defines the number of running real-time applications.
**.ecu1.numApps = 3

# Define tagged streams
**.ecu1.phy[*].taggedVIDs = "0,1"

#App: ----- AVB Listener -----
# receiving data from Camera/Radar
**.ecu1.app[0].typename = "AVBTrafficSinkApp"
**.ecu1.app[0].streamID = 0

#App: ----- AVB Listener -----
# receiving data from Camera/Radar
**.ecu1.app[1].typename = "AVBTrafficSinkApp"
**.ecu1.app[1].streamID = 1
```

```
# App: ----- TT-Traffic -----
# Defines the running real-time application that generates TT traffic.
# TT control traffic for steering actuator
**.ecu1.app[5].typename = "TTTrafficSourceApp"
**.ecu1.app[5].displayName = "vl_101"
**.ecu1.app[5].action_time = 996us
**.ecu1.app[5].payload = 100Byte
**.ecu1.app[5].ct_id = 101

# Connect the traffic generator output to the corresponding buffer.
**.ecu1.app[5].buffers = "vl_101"

# Initialization of second buffer windows
**.ecu1.vl_101.ctc.receive_window_start = sec_to_tick(995us)
**.ecu1.vl_101.ctc.receive_window_end = sec_to_tick(1015us)
**.ecu1.vl_101.ctc.permanence_pit = sec_to_tick(1015us)
# Connect buffer of vl_101 to phy[0]
**.ecu1.vl_101.destination_gates = "phy[0].TTin"
**.ecu1.vl_101.ct_id = 101
**.ecu1.vl_101.sendWindowStart = sec_to_tick(1025us)

# Register the buffers of vl_101 as TT output traffic phy[0]
# This reserves the slots required for the corresponding TT messages
**.ecu1.phy[0].shaper.tt_buffers = "vl_101"

# Initialization of first buffer windows
**.ecu1.vl_100.ctc.receive_window_start = 0
**.ecu1.vl_100.ctc.receive_window_end = 0
**.ecu1.vl_100.ctc.permanence_pit = 0
**.ecu1.vl_100.sendWindowStart = 0
**.ecu1.vl_100.ct_id = 0
```

Listing 5.7: Code Example for Central Processing ECU HW Initialization

In practice, the .ini initialization file specifies all the parameters required for the simulation. These parameters include the physical MAC address, the number of applications running on the device, and the configuration details for how these applications send or receive data, including payloads and transmission windows (when necessary). These files facilitate the configuration of the simulation. In fact, for example, if a different payload needs to be simulated, it is possible to only change it without rewriting all the lines of code.

Finally, to complete the range of configuration examples, we provide the definition

of the .ini file of an actuator receiving TTEthernet frames.

We remember that TT traffic is essential for sending commands or receiving signals that require deterministic latency for critical applications, such as drive-by-wire systems.

The TT actuator can be configured through an initialization file as follows:

```
# Network name
network = SIMTTAVBO

# MAC address definition
**.sbwFrontLeft.phy[*].mac.address = "0C-10-00-00-00-01"
# Defines the number of running real-time applications.
**.sbwFrontLeft.numApps = 1

# Listener TT-Traffic
**.sbwFrontLeft.app[0].typename = "CTTrafficSinkApp"
**.sbwFrontLeft.app[0].displayName = "vl_101"

# Connect the conformance checker for incoming TT traffic
# of vl_101 with phy[0] such that the corresponding
# TT messages will be put into the buffer at ppt.
**.sbwFrontLeft.phy[0].inControl.ct_incomings = "vl_101_ctc"
**.sbwFrontLeft.vl_101.destination_gates = "app[0].TTin"
**.sbwFrontLeft.vl_101.ct_id = 101
**.sbwFrontLeft.vl_101_ctc.receive_window_start = sec_to_tick(1070us)
**.sbwFrontLeft.vl_101_ctc.receive_window_end = sec_to_tick(1210us)
**.sbwFrontLeft.vl_101_ctc.permanence_pit = sec_to_tick(1210us)
**.sbwFrontLeft.vl_101.sendWindowStart = sec_to_tick(1310us)
```

Listing 5.8: Code Example for Actuator HW Initialization

5.3.2 TTAVBEthernetSwitch

We use the *TTAVBEthernetSwitch* module to model both the zonal switches, which collect and forward data to and from sensors and actuators located in different vehicle zones, and the central switch, which forwards data between the zonal switches and the central processing ECU.

The central switch, in addition to forward data from sensors and to actuators, also forwards Ethernet packets carrying CAN messages converted from the CAN domain through the CAN-Ethernet gateway. An example of the modeled hardware architecture is shown in Figure. 5.5.

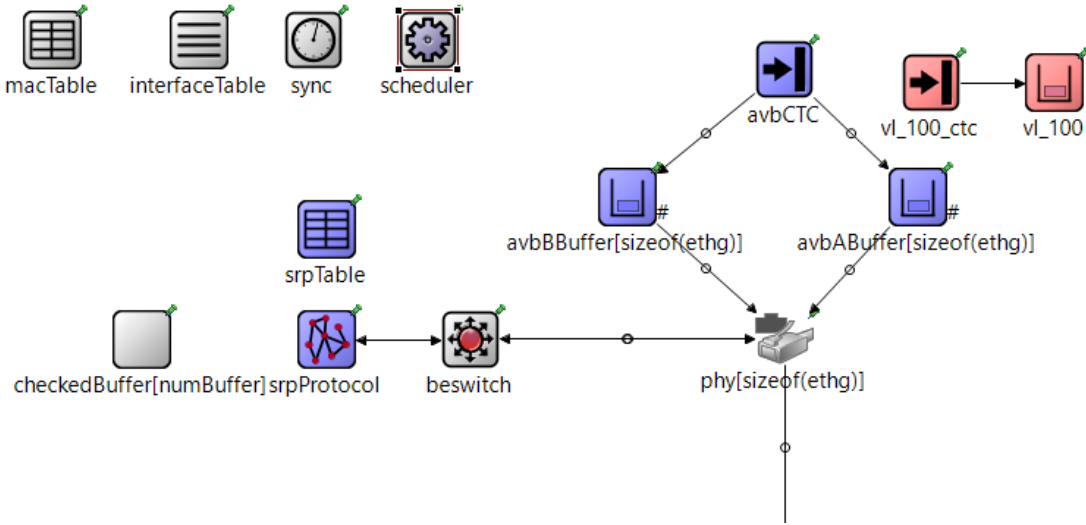


Figure 5.5: TTAVBEthernet Switch .ned Model (GUI)

While a general structure for the .ned file can be coded as follows:

```
package signalsandgateways.examples.SIMTTAVB0;

import core4inet.nodes.ethernet.avb_AS6802.TTEAVBEtherSwitch;
import core4inet.incoming.AS6802.TTIncoming;
import core4inet.buffer.AS6802.TTDoubleBuffer;

//configuration in switchMiddleLeft.ini.
module Switchmiddleleft extends TTEAVBEtherSwitch
{
    submodules:
        vl_100_ctc: TTIncoming {
            parameters:
        }
        vl_100: TTDoubleBuffer {
            parameters:
        }
    connections:
        vl_100_ctc.out --> vl_100.in;
}
```

Listing 5.9: Code Example for Switch HW Configuration

Some of the components are the same as the TTEthernetHost. The different components include:

- **Switch (beswitch):** This module forwards frames to the appropriate ports based

on their destination MAC addresses. Additionally, it handles AVB SRP frames. Incoming SRP frames are forwarded as SRP messages through the gate connected to the `srpProtocol` module, while outgoing SRP messages are forwarded as EtherFrames to the physical port (`phy`) according to the control information.

- **Switch TT Buffer (`checkedBuffer`):** Module for buffer handling, including a module for conformance checking of incoming TT (Time-Triggered) traffic.
- **MAC Table (`macTable`):** Interface for MAC address tables.

While a general structure for the `.ini` file can be coded as follows:

```
# Define MAC Addresses
**.switchMiddleLeft.phy[0].mac.address = "0B-02-00-00-00-01"
**.switchMiddleLeft.phy[1].mac.address = "0B-02-00-00-00-02"

# Define PHY IF properties
**.switchMiddleLeft.phy[0].inControl.ct_incomings = "vl_100_ctc"
**.switchMiddleLeft.phy[1].shaper.tt_buffers = "vl_100"
**.switchMiddleLeft.phy[1].shaper.AVBHigherPriorityBandwidth = 3Mbps
**.switchMiddleLeft.phy[2].inControl.ct_incomings = "vl_100_ctc"

**.switchMiddleLeft.vl_100_ctc.receive_window_start = sec_to_tick
(995.120us)
**.switchMiddleLeft.vl_100_ctc.receive_window_end = sec_to_tick
(1015.120us)
**.switchMiddleLeft.vl_100_ctc.permanence_pit = sec_to_tick(1015.120us
)
**.switchMiddleLeft.vl_100.destination_gates = "phy[1].TTin"
**.switchMiddleLeft.vl_100.ct_id = 100
**.switchMiddleLeft.vl_100.sendWindowStart = sec_to_tick(1024.960us)
```

Listing 5.10: Code Example for Switch HW Initialization

The number of `phy` ports depends on devices connected to the switch, and configuration may depend on the application requirements.

5.3.3 BusEthernetGateway

The module *BusEthernetGateway* allows to model a gateway able to convert CAN bus messages to Ethernet frames and vice versa. An example of the modeled hardware architecture is shown in Figure. 5.6.

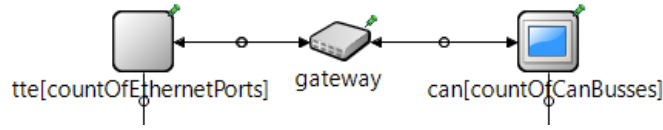


Figure 5.6: TTAVBEthernet Gateway .ned Model (GUI)

In the .ini must be specified the ID of the incoming CAN frames. The gateway design is specified through a .xml file, as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--<!DOCTYPE config SYSTEM "../src/signalsandgateways/gateway/
  GatewayConfig.dtd"-->
<config>
  <gateway id="gateway1">
    <routing>
      <route source="0" destination="1">
        <canframe canId="419348974"/>
        <canframe canId="419358446"/>
      </route>
    </routing>
    <buffering>
      <pool>
        <holdup time="0ms">
          <canframe canId="419348974"/>
        </holdup>
        <holdup time="0ms">
          <canframe canId="419358446"/>
        </holdup>
      </pool>
    </buffering>
    <transformation destination="1" type="toEthernet">
      <ethernetframe dst="0D-00-00-00-00-01">
        <canframe canId="419348974"/>
        <canframe canId="419358446"/>
      </ethernetframe>
    </transformation>
  </gateway>
</config>
```

Listing 5.11: Code Example for Gateway Design

The structure is simple and it is based on three main components:

- **Ethernet Interface (tte)**: Interface for Ethernet hosts in the gateway.
- **Gateway (gateway)**: A basic gateway that forwards messages from the source to the desired destination.
- **CAN Interface (can)**: Represents the CAN port of a gateway.

Two CAN messages are forwarded within the Ethernet domain, with the destination MAC address specified in the file. The .xml file allows for specifying a Hold-Up value. Since the minimum Ethernet payload is 46 bytes and the standard CAN message payload is 8 bytes, directly forwarding CAN data results in Ethernet payload padding, wasting bandwidth [58]. Given a Hold-Up time, the gateway can perform CAN aggregation, releasing frames in groups to the Ethernet domain. More messages are assigned to a shared pool and transmitted through Ethernet when the Hold-Up time expires. The drawback is an increase in latency and jitter. A Hold-Up time of 0 means CAN messages are immediately forwarded without any additional delay.

5.3.4 Complete Network Architecture Simulation

All these modules, which define specific Hardware (HW) and Software (SW) features of the components, can be combined to form a complete network simulation using the .ned general file. An example of a complete network is shown in Figure 5.7.

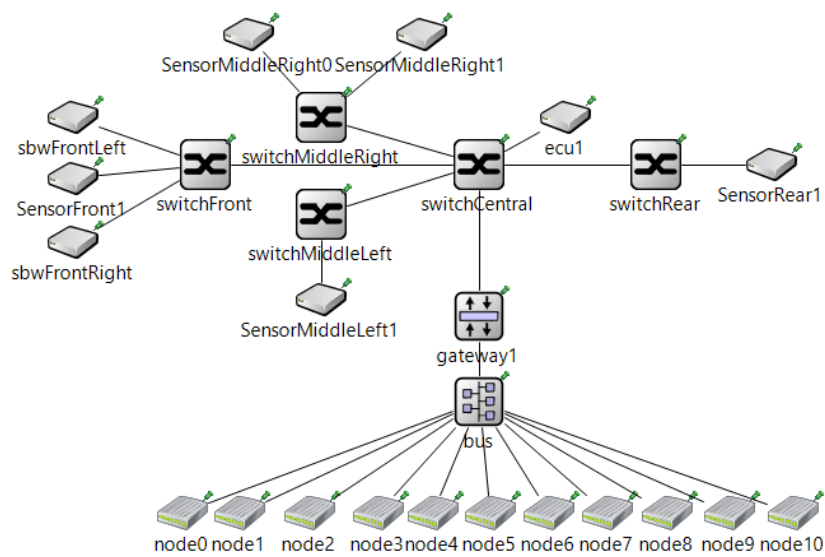


Figure 5.7: GUI Example for Full Network Configuration

In the .ned general files, it is essential to include and instantiate all components used in the network. Each component connection must be explicitly specified. Additionally, parameters such as the number of ports for each device, the data rate of the links, and cable delays must be defined. An example is provided below:

```
package signalsandgateways.examples.SIMTTAVB0;
import core4inet.nodes.ethernet.AS6802.TTEtherHost;
import signalsandgateways.BusEthernetGateway;
import fico4omnet.bus.can.CanBus;
import fico4omnet.nodes.can.CanNode;

network SIMTTAVB0
{
    parameters:
        types:
            channel cable extends ned.DatarateChannel
            {
                datarate = 100Mbps;    // Definition of Ethernet cable
                ber = 0;
                per = 0;
            }
        submodules:
            switchFront: Switchfront {
                parameters:
                gates:
                    ethg[4];
            }

            switchMiddleLeft: Switchmiddleleft {
                parameters:
                gates:
                    ethg[2];
            }

            switchMiddleRight: Switchmiddleright {
                parameters:
                gates:
                    ethg[3];
            }

            switchRear: Switchrear {
                parameters:
                gates:
```

```

        ethg[2];
    }

    switchCentral: Switchcentral {
        parameters:
        gates:
            ethg[6];
    }

    SensorFront1: Sensorfront1 {}
    SensorMiddleLeft1: Sensormiddleleft1 {}
    SensorMiddleRight0: Sensormiddleright0 {}
    SensorMiddleRight1: Sensormiddleright1 {}
    SensorRear1: Sensorrear1 {}
    ecu1: Ecu1 {}
    sbwFrontLeft: Sbwfrontleft {}
    sbwFrontRight: Sbwfrontright {}

    bus: CanBus {
        gates:
            gate[12];
    }

    node0: CanNode {}
    node1: CanNode {}
    node2: CanNode {}
    node3: CanNode {}
    node4: CanNode {}
    node5: CanNode {}
    node6: CanNode {}
    node7: CanNode {}
    node8: CanNode {}
    node9: CanNode {}
    node10: CanNode {}

    gateway1: Gateway1 {
        gates:
            cang[1];
            ethg[1];
    }

connections:
    SensorFront1.ethg <--> cable { delay = 50ns; } <-->

```

```

switchFront.ethg[0];
    switchCentral.ethg[0] <--> cable { delay = 50ns; } <-->
switchFront.ethg[1];
    SensorMiddleLeft1.ethg <--> cable { delay = 50ns; } <-->
switchMiddleLeft.ethg[0];
    switchCentral.ethg[1] <--> cable { delay = 50ns; } <-->
switchMiddleLeft.ethg[1];
    SensorMiddleRight0.ethg <--> cable { delay = 50ns; } <-->
switchMiddleRight.ethg[0];
    SensorMiddleRight1.ethg <--> cable { delay = 50ns; } <-->
switchMiddleRight.ethg[1];
    switchCentral.ethg[2] <--> cable { delay = 50ns; } <-->
switchMiddleRight.ethg[2];
    SensorRear1.ethg <--> cable { delay = 50ns; } <--> switchRear
.ethg[0];
    switchCentral.ethg[3] <--> cable { delay = 50ns; } <-->
switchRear.ethg[1];
    ecu1.ethg <--> cable { delay = 50ns; } <--> switchCentral.
ethg[4];
    bus.gate[0] <--> node0.gate;
    bus.gate[1] <--> node1.gate;
    bus.gate[2] <--> node2.gate;
    bus.gate[3] <--> node3.gate;
    bus.gate[4] <--> node4.gate;
    bus.gate[5] <--> node5.gate;
    bus.gate[6] <--> node6.gate;
    bus.gate[7] <--> node7.gate;
    bus.gate[8] <--> node8.gate;
    bus.gate[9] <--> node9.gate;
    bus.gate[10] <--> node10.gate;
    bus.gate[11] <--> gateway1.cang[0];
    switchCentral.ethg[5] <--> cable { delay = 50ns; } <-->
gateway1.ethg[0];
    switchFront.ethg[2] <--> cable { delay = 50ns; } <-->
sbwFrontLeft.ethg;
    switchFront.ethg[3] <--> cable { delay = 50ns; } <-->
sbwFrontRight.ethg;
}

```

Listing 5.12: Code Example for Full Network Configuration

5.4 Batch Interface

As anticipated, our simulations are run inside a MinGW command line interface within a specific folder that acts as a virtual environment. This environment contains all the necessary configurations to manage the simulations correctly. To handle the interface between the optimizer and the simulation environment, batch Application Programming Interfaces (APIs) were developed. These APIs simplify the process and ensure that the software remains independent of the simulator, making it easy to extend to other simulation environments in the future.

Simulations are executed using a bash script in OMNeT++ without a graphical user interface to save resources. The software automatically communicates with the `runner.bash` script by writing and clearing flags that signal the start and end of the simulation. This setup allows for efficient and automated management of the simulation process. A conceptual graph explaining how the interface is used to manage the simulation is shown in Figure 5.8.

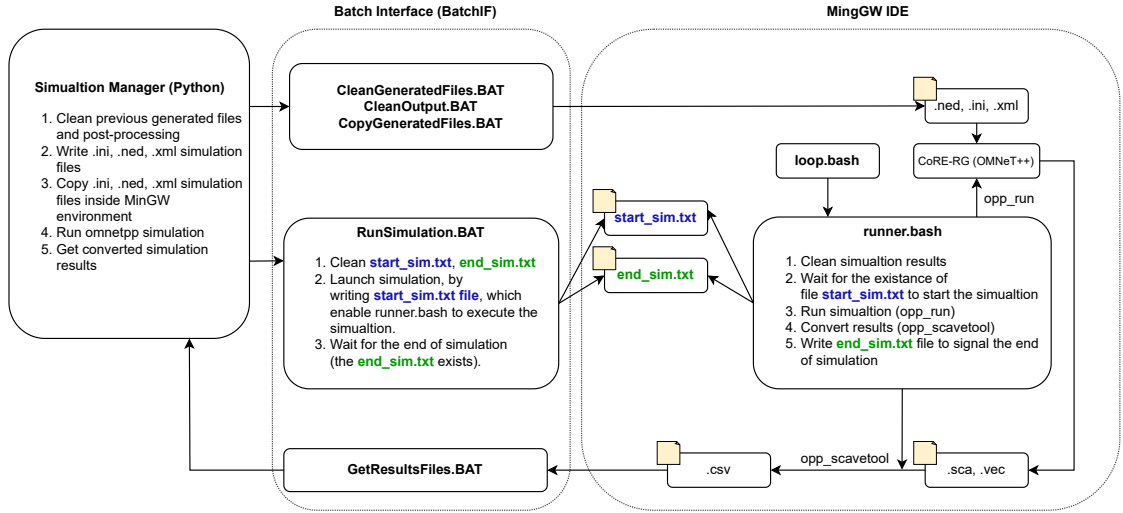


Figure 5.8: BatchIF

Before running the automated design software, the simulation environment must be opened through the MinGW command line, and the scripts that wait for a call to the simulation must be started. Particularly, this is done using two bash scripts, as operations in the virtual simulation environment must be performed through bash scripting language. The first script (`loop.bash`) acts as a wrapper for the second (`runner.bash`), which handles all necessary operations to clean data, start the simulation, and convert the results.

The Python software, before starting the simulation, cleans any previously generated files and artifacts related to post-processing (*CleanGeneratedFiles.BAT*, *CleanOutput.BAT*). This ensures that all codes are related to the current optimization and no old artifacts affect the final results. A new set of configuration files is then generated from the optimizer resulting architecture and copied (via *CopyGeneratedFiles.BAT*) into the simulation environment. The simulation is started, and results are converted and retrieved (via *getResultsFiles.BAT*) for post-processing operations. All these tasks are managed through simple batch scripts.

The main part of this interface involves the *RunSimulation.BAT* batch script that runs the simulation. This script generates a file to signal the start of the simulation ('start_sim.txt'), which is read by the *runner.bash* script inside the simulation environment. When the simulation is finished and data conversion is completed, the *runner.bash* script writes a file ('end_sim.txt') to signal the end of the simulation, which is read by the *RunSimulation.BAT* batch script.

Since the Python calls to these batch scripts are sequential, meaning no other operations are performed until a batch script has finished its execution, this method allows for communication between the Python software and the simulation environment. They effectively run in parallel, with the start and end of the simulation signaled via simple text files.

Chapter 6

Evaluation and Results

In this chapter, we present the outcomes of our research on design space exploration methodology.

We start by detailing the initial development of our framework, where optimization is automated, and simulations are manually validated.

We then introduce the concept of integrating simulation and optimization, which involves the automatic generation and retrieval of simulation files. However, this initial approach lacked a method for regenerating problem constraints when solutions were infeasible and did not automate the optimization and simulation loop. We applied this concept to a different domain, specifically indoor robotic networks design. This demonstrated that, although initially focused on off-road vehicles, our results can be generalized to other areas of interest.

Finally, we completed the framework by integrating both techniques, reformulating constraints, and implementing an iterative mechanism to fully automate the design procedure. We used this framework to design a comprehensive E/E system, considering all necessary requirements. Our evaluation includes Pareto fronts, trade-off analysis, and a qualitative scalability analysis to understand how the framework performs in larger future scenarios.

Since the backgrounds of the first and third evaluation steps have already been extensively discussed in the introductory chapters of this dissertation, we will provide an overview of the reference sector only for the second evaluation phase, which focuses on industrial robotics.

6.1 DSE for Off-Road Vehicles and GSR Requirements

In this section, we present the results of our initial evaluation. The primary objective is to establish an optimization environment tailored for the E/E architecture of off-road vehicles, with a particular focus on supporting GSR/ADAS functionalities. Although the optimization design process is automated, the simulation files and results analysis remain manual tasks. This progress is further elaborated in the sections discussing our subsequent evaluations.

The level of automation within the framework, as aligned with the adopted methodology, is depicted in Figure 6.1.

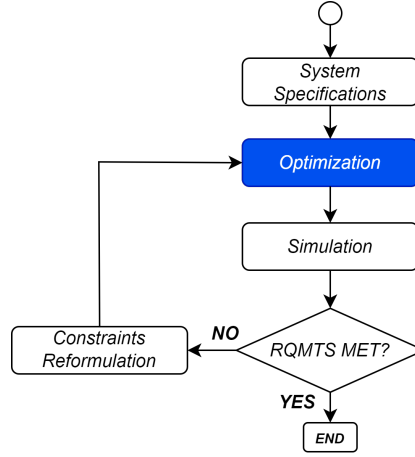


Figure 6.1: Level of Development of the Framework in the Current Evaluation: The optimization process is fully automated, while simulation and constraint reformulation are fulfilled manually.

Beyond its practical implications, this evaluation step serves as a baseline to validate the optimization component of the framework. It demonstrates whether maximization and minimization problems are correctly solved, providing sensible and feasible solutions.

6.1.1 Problem Definition

The framework inputs include the problem file, which contains the problem template, constraints, and connection rules among components, and the library file, which contains the hardware components (i.e., real components) with their features.

The problem template is a redundant zone-based architecture divided into four vehicle zones: front-left, front-right, rear-left, and rear-right. The components used in the template include sensors (automotive Radar or Cameras), zonal Ethernet switches,

a central Ethernet switch, and a central processing ECU (ADCU). The central Ethernet switch provides the backbone link between the zonal switches and the central processing ECU.

The problem constraints include maximum system cost, power consumption and weight, which are fixed during the design phase due to budgetary, mechanical, and electrical design constraints.

Connection rules are used to restrict certain connections among hardware components (e.g., a sensor must be connected to the central processing ECU via switches) and to add logical rules to the system topology.

The problem library is a collection of off-the-shelf components available in the market, with their cost, rate, power consumption, and weight.

Since it is technically infeasible to simulate all possible system architectural combinations due to time constraints, optimization is used to select the best implementations among a set of solutions, which are then validated through simulation. Specifically, we validate the results in terms of latency, jitter, and bandwidth occupation (i.e., network load).

The optimizer takes the problem file and the library file as inputs and provides the optimal system topology and mapping to real components that maximize (or minimize) a given cost function, considering specific design constraints such as weight, cost, and power consumption.

After converting the input files into MILP numerical formulation, the CPLEX optimizer is employed to solve the problem. This process results in the assignment of binary variables on the adjacency matrix (representing the system topology) and the mapping matrices (representing the system implementation).

6.1.2 Optimization Design Results

In the current work, two optimization goals are considered. The first is to maximize the data rate to collect the most information from the sensors, while the second is to minimize the bandwidth utilization on the backbone to allow for future system extensions with additional sensors, enhancing the architecture scalability. Through these scenarios, we also test the optimization tool to ensure that the MILP formulation is correctly implemented and leads to feasible and accurate solutions.

In Figure 6.2, the system architecture and mapping results (for sensors) are reported.

In both optimization scenarios, only 4 out of the 16 available sensors are selected.

Each sensor is correctly connected to the central processing unit through a zonal

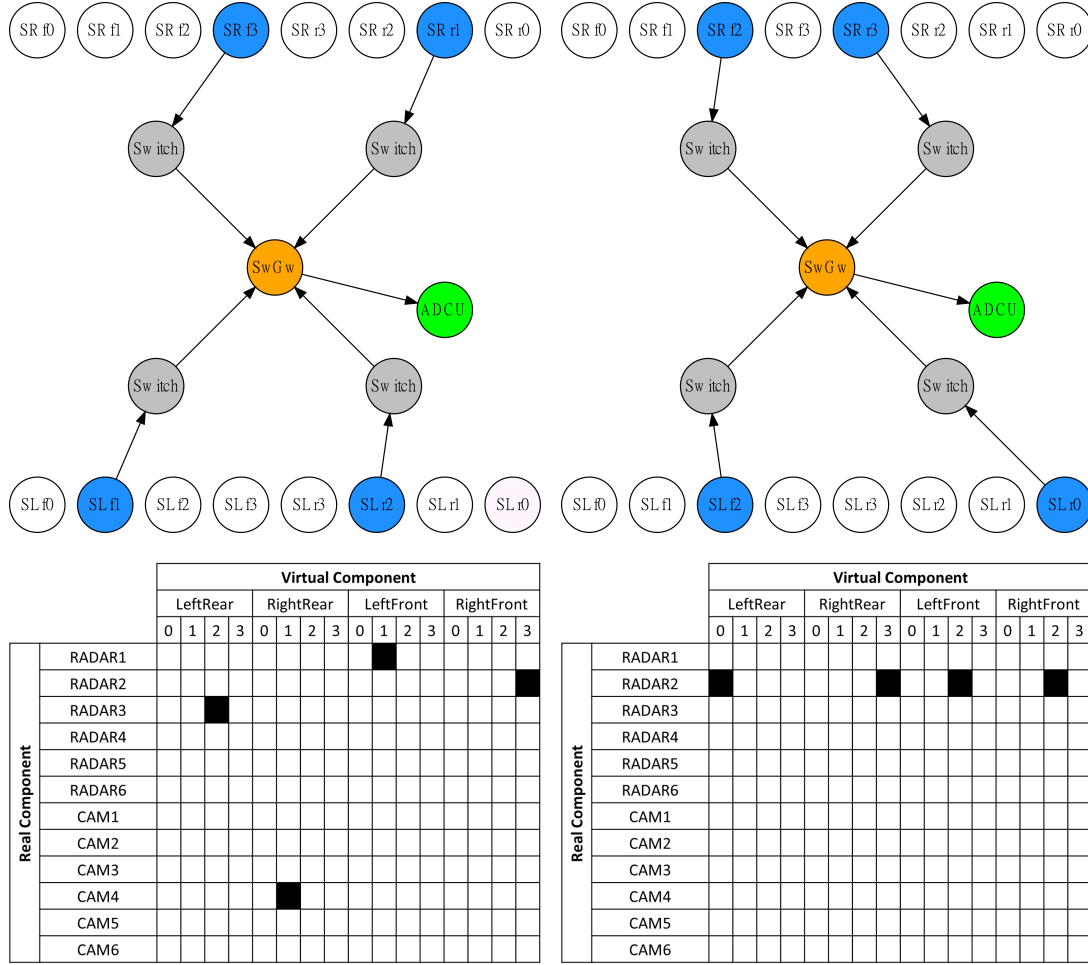


Figure 6.2: System Architecture and Mapping Optimization Results for Maximizing Data Rate (Left) and Minimizing Bandwidth Occupation at Backbone (Right).

switch and a central switch, ensuring that components of the same type are not connected to each other. This indicates that the connection rules and path constraints are accurately modeled and implemented.

Components that are not instantiated do not have a mapping, while instantiated nodes are implemented using one and only one hardware component available in the market out of 12 options. This confirms that the mapping constraints are also correctly modeled and implemented within the framework.

Regarding problem constraints (reported in Table 6.1), in both scenarios the total system cost, weight, and power consumption are all within the specified limits, demonstrating that these constraints are accurately modeled in the tool.

In the first optimization scenario, which focuses on maximizing data rate, the configuration includes three radars and one camera. Conversely, in the second optimization

Table 6.1: Total System Cost, Power Consumption and Weight for Maximizing Data Rate (Scenario 1) and Minimizing Bandwidth Occupation at Backbone (Scenario 2).

	Cost [EU]	Power Consumption [W]	Weight [kg]
Scenario 1	6450	18,3	15.7
Scenario 2	5300	18,2	14.7
Design Constraint	6800	19	16.5

scenario, aimed at minimizing bandwidth utilization, only radars are employed. This outcome is anticipated, as the data rate of the camera is significantly higher than that of the radars. This also confirms that the maximization and minimization mechanisms are correctly modeled and implemented.

6.1.3 Design Results Validation

After obtaining the design results, which include the architecture and the mapping between virtual and real hardware components, we validated the architectural outcomes of both optimization scenarios.

During this phase, the simulation files (i.e., .ned, .ini, and .xml files) were manually created. In addition to the optimized architectural components related to ADAS functions, the current/legacy architecture based on CAN, which is used for standard vehicle functions (ABS, ECM, etc.), was included in the simulation environment. This integration allowed us to test the compatibility of the new optimized ADAS components with the existing CAN network. To facilitate communication between the CAN and Ethernet domains, a gateway was configured to forward the necessary vehicular CAN messages (such as Vehicle Speed, Gear Information, Transfer Case Status, PTO Status, and Differential Lock Status) to the central processing ECU.

After running the simulation, results data were manually extracted, analyzed and plotted, taking a window of 5 seconds to appreciate latency and jitter behaviors.

In Figure 6.3 the latency of the two optimized scenarios, for all streams, is shown.

In the first scenario, the latency over time for the camera, which transmits more data ('Stream 3'), is clearly distinguishable. In contrast, the data transmitted by the radars exhibit more consistent behavior across both scenarios.

In both scenarios, since AVB Class-A streams are transmitted from the sensors and received by the central ECU, latency varies over time due to the credit-based shaper algorithm. Stream packets are held in queues until the credit is non-negative, causing latency to fluctuate, even though it is guaranteed within a 2 ms bound. In

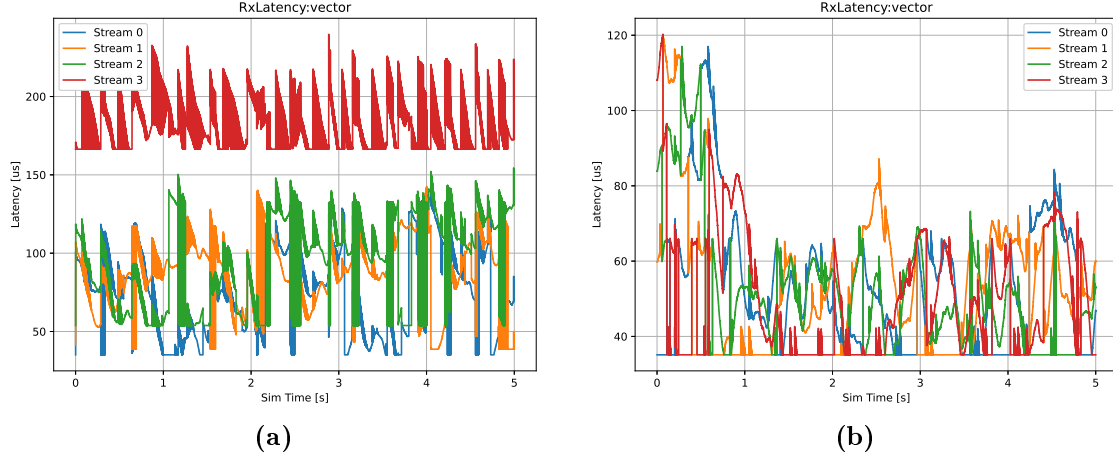


Figure 6.3: Simulated Network Latency Results for Maximizing Data Rate (a) and Minimizing Bandwidth Occupation at Backbone (b).

this application, where the central ECU acts as a sink and the sensors are only sources, this variability is not problematic. However, for applications requiring control, such as drive-by-wire systems, this type of traffic is unsuitable because the latency, while bounded, is not constant and always deterministic. In such cases, other communication standards like TTEthernet would be more appropriate.

In Figure 6.4 the latency of the two optimized scenarios, for all streams, is shown.

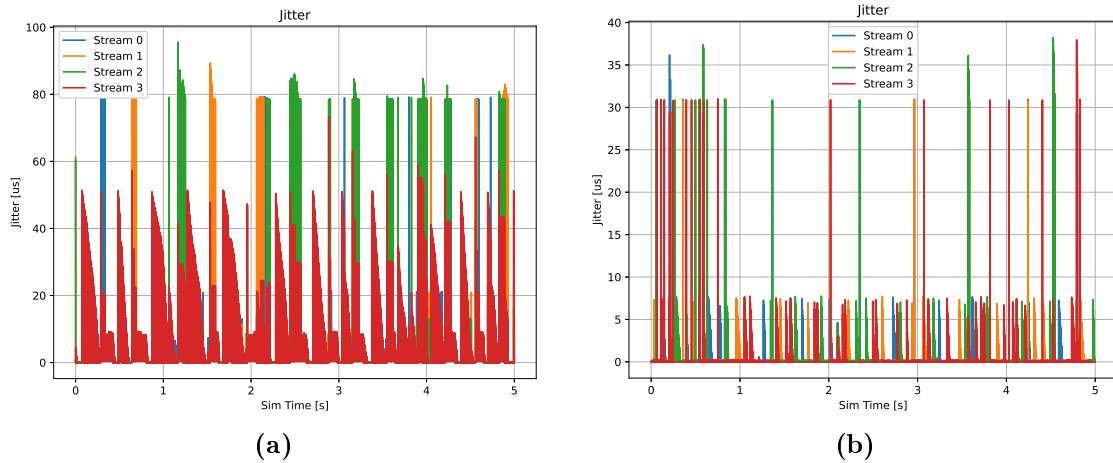


Figure 6.4: Simulated Network Jitter Results for Maximizing Data Rate (a) and Minimizing Bandwidth Occupation at Backbone (b).

It is possible to see that the jitter, which measures the variability in packet arrival times over a network, tends to increase when more bandwidth is consumed. This is

because higher bandwidth usage leads to more traffic in the hardware queues, resulting in greater variability in packet arrival times.

In the first scenario, the increased bandwidth usage causes all streams to experience higher jitter. This is due to the increased number of packets accumulating in the hardware queues, which leads to more inconsistent packet delivery times.

The spikes in jitter observed in the graphs are caused by CAN messages being converted to Ethernet frames, representing the cross-traffic in the network.

Table 6.2 presents the precise values for both scenarios of latency and jitter, including the mean and maximum values for each stream.

Table 6.2: Mean and Maximum Values for Latency and Jitter for Scenario 1 (Sim 1) and Scenario 2 (Sim 2).

	Sensor	BW [Mbps]	SR Class	Stream ID	Mean Latency [μ s]	Max Latency [μ s]	Mean Jitter [μ s]	Max Jitter [μ s]
Sim 1	Radar	5	SR-A	1	77.62	141.07	1.25	84.79
	Radar	6	SR-A	2	89.96	142.60	1.29	89.38
	Radar	10	SR-A	3	97.31	154.43	1.71	95.61
	Camera	40	SR-A	4	175.43	239.72	2.28	73.33
Sim 2	Radar	5	SR-A	1	53.30	117.03	0.11	36.18
	Radar	5	SR-A	2	53.51	120.25	0.11	30.99
	Radar	5	SR-A	3	51.42	117.03	0.12	38.25
	Radar	5	SR-A	4	49.98	120.22	0.11	37.97

Considering the bus load in the CAN network, we conducted two simulations: one using the legacy architecture and another incorporating an additional ECU for the Central Tyre Inflation System (CTIS) and additional warning messages to meet GSR requirements. The ECU is responsible for monitoring tire pressure and managing the inflation/deflation of the tires. The warning messages are used to alert the driver when specific vehicle conditions occur.

Results of CAN and Ethernet network load (measured at the backbone) are presented in Table 6.3.

Table 6.3: CAN and Ethernet Network Load for Scenario 1 (Sim 1) and Scenario 2 (Sim 2).

	Calculated Bus Load [%]	Simulated Bus Load [%]	BW occupation at Backbone [%]
Legacy Architecture	35.82	34.10	Not Applicable
Scenario 1 (Sim 1)	48.91	48.02	61
Scenario 2 (Sim 2)	48.91	48.02	20

Regarding the CAN bus load, the addition of the ECU and warning messages to meet GSR requirements results in an approximate 13% increase in network load. This increase is consistent across both scenarios, as this part of the network remains fixed while the optimizer adjusts the design of the Ethernet domain.

In the Ethernet domain, the bandwidth occupation measured at the backbone is 61% in the first scenario. This is expected since the goal was to maximize the data rate. However, it is below the 75% threshold suggested by AVB, as higher values often lead to infeasible solutions due to cross-traffic and other protocol mechanisms like the stream reservation protocol, which cannot allocate all streams at higher loads. This underscores the challenge of modeling certain behaviors with the optimizer alone, and highlights the importance of simulation in refining results to help the optimizer converge on a practically feasible solution.

In the second scenario, the bandwidth occupation at the backbone is measured at 20%. This outcome is anticipated, as the optimization goal was to minimize the data rate to allow for future architectural extensions with additional sensors. These results confirm that the optimizer is functioning as intended.

Regarding the validation of all the analyzed data, the latency requirements for each stream are satisfied in both scenarios. Specifically, the latency for each data stream is significantly lower than the 2 ms strict latency constraint upper bound defined for class D SAE classes [32].

Considering jitter, in this design case, which focuses solely on forwarding data from sensors to the central processing unit, it does not have significant implications. However, the discussion changes when time-sensitive tasks must be met, where jitter generally needs to be close to zero [53]. This can be achieved through scheduling improvements in AVB or by using other standards like TTEthernet, as we will demonstrate in other works.

Finally, the bus load increase of 13% (from 34.10% to 48.02%) remains within the 50% limit to ensure that all CAN messages are transmitted according to their transmission periods [47].

6.1.4 Discussion

In this preliminary evaluation, aimed at providing an initial validation of the optimization component of our framework, we applied our design concepts to a real-world scenario involving the E/E system architecture for off-road vehicles.

Regarding problem constraints, we modeled system cost, power consumption, data rate, and weight through the optimizer, demonstrating that it leads to correct solutions.

Other network metrics, such as network load, latency, and jitter, were not modeled through the optimizer but were measured and evaluated through simulation. An initial finding from this approach is that modeling only the data rate with the optimizer can

lead to infeasible solutions. Conversely, modeling more precise protocols and communication mechanisms, such as the stream reservation protocol for AVB, scheduling, and collision, is very complex or nearly impossible with a MILP-based optimizer. Therefore, we rely on simulation to capture these network behaviors.

Although in this first evaluation step we manually created simulation files and evaluated the results, in the next evaluation we will automate these tasks. In fact, our goal is to develop a fully autonomous framework that iteratively uses both optimization and simulation to converge on a final system design solution.

6.2 DSE for Indoor Robotics Networks

In this section, we present our second evaluation, which aims to enhance the automation level of our design space exploration framework by automating the generation of simulation files, and the retrieval and analysis of results. Additionally, we introduce the concept of an iterative technique that combines both optimization and simulation, which will be fully described in our third evaluation, thereby completing our framework.

The level of automation within the framework, as aligned with the adopted methodology in the current evaluation, is depicted in Figure 6.5.

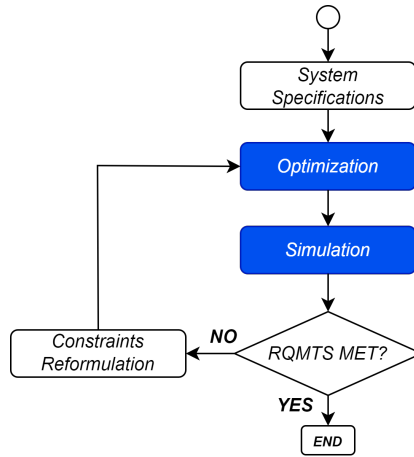


Figure 6.5: Level of Development of the Framework in the Current Evaluation: The optimization and simulation process is fully automated, while constraint reformulation is conceptually introduced

To test this part of the framework, we focused on a different domain: the design of indoor robotic networks. Unlike our first and third evaluations, which are oriented towards the off-road vehicle domain as discussed in the introduction of this dissertation, this section also provides a background on the indoor robotics network domain to enhance understanding of the evaluation. We also provide implementation details of the optimizer, which enable us to address this specific problem effectively.

6.2.1 Background

The ongoing development objective of 5G wireless communication is to achieve ubiquitous connectivity for a wide range of user devices and applications, including autonomous guided vehicles, unmanned aerial vehicles, industrial automation, and augmented and virtual reality.

Emerging Ultra-Reliable and Low-Latency Communication (URLLC) applications, such as autonomous robots and industrial automation, are both computation-intensive and delay-sensitive. These applications impose diverse Quality of Service (QoS) requirements, particularly in terms of high reliability and low latency [20]. The Third Generation Partnership Project (3GPP) has established the latency and reliability requirements for URLLC applications at 1 ms and 99.999%, respectively [49], as highlighted in Table 6.4 [1].

Table 6.4: Expected QoS RQMTs for URLLC applications

Industry	E2E Delay [ms]	Reliability
Internet of Things	1	10^{-5}
Industry Automation	1	10^{-5} – 10^{-9}
Autonomous Guided Vehicles	5–10	$\geq 10^{-3}$
Virtual reality	5–10	10^{-3} – 10^{-5}

In this context, effective task offloading management is crucial for delivering high-quality services [71], necessitating optimal management of computing and communication resources at the network edge. With the significant increase in network density and heterogeneity, and to enhance both system QoS and user Quality of Experience (QoE), emerging infrastructures and tools like multi-access edge computing (MEC) are highly desirable for network providers to optimally deploy low-latency computational resources at the network edge.

The literature presents several optimization approaches focusing on both static scenarios [90, 15] and applications that must explicitly account for user mobility [112, 69, 16, 73]. These approaches propose optimization algorithms or iterative procedures to optimize the allocation of servers to tasks while respecting energy, latency, and communication delay constraints.

However, most of the literature is concerned with optimizing a given architecture. In this work, we focus on the earlier phase of architectural design: the placement of infrastructure, the deployment and sizing of servers, and the distribution of communication and services. Our objective is to automatically generate the architecture from a relatively simple description of the environment and application requirements, expressed as performance and quality of service constraints.

Task offloading in mobile scenarios is indeed a complex challenge. As mobile users move across network attachment points, they must be able to connect to edge servers that can meet the expected service levels.

In a relevant work [73], the resource allocation and optimization problem for a given architecture in a mobility-based industrial robotics scenario was tackled. A mixed-integer linear programming (MILP) formulation to model service migration and to maximize the overall network QoS, including communication, computation, and latency, was used.

More recently, preliminary results on using design-space exploration techniques to *construct* an optimized architecture and its resources based on high-level requirements [86] was presented. The work, based on the ArchEx framework [63], employs a pattern-based language to simplify problem formulation and mixed-integer linear constraints to solve both allocation and mapping problems.

However, several aspects of real deployments are either too complex to model for the optimizer or lead to excessive run-times due to the combinatorial nature of the optimization problem. Simulation, on the other hand, can easily handle details such as protocol messages and run-time collisions but does not scale well to large design spaces. Therefore, our approach combines these two techniques.

For example, Finn et al. link an optimization algorithm that generates candidate architectures for an aircraft environmental control system over a discrete space with the MODELICA simulator to perform analysis over a continuous space [24].

Similarly, Moin et al. synthesize topologies for body area networks (BANs) under energy and reliability constraints using an iterative in-the-loop optimization technique based on a discrete-event network simulator [60].

In this evaluation, we adopt a similar approach by complementing a symbolic MILP optimization strategy with a simulation environment. This allows us to estimate several key aspects that would otherwise be too costly to handle due to lower-level details.

Specifically, we integrate ArchEx [63] with the OMNeT++ discrete-event simulator [98], enabling the automatic generation of simulation configuration files. This integration is potentially extendable for iterative optimizations.

Our idea significantly enhances design automation by allowing the optimizer to generate candidate solutions through various patterns, with simulation results being automatically retrieved.

6.2.2 Problem Formulation

We formalize the exploration as a MILP optimization problem, adopting the baseline methodology of ArchEx. We extend the tool structure by re-creating the basis constraints and implementing a new group of application constraints. Here, we briefly

recall the formulation and refer the reader to our previous work for details [63]. The architecture model, or *template*, is a directed graph $\mathcal{G} = (\mathcal{V}, E)$, which represents a network of interconnected components. The edges $e_{ij} \in E$ of $\mathcal{G}$ are binary variables representing a connection between nodes v_i and v_j , where $e_{ij} = 1$ or 0 indicates the presence or absence of a connection between the source and destination. Similarly, a node v_i is represented by a binary variable δ_i , which evaluates to 1 if at least one incoming edge e_{ji} or outgoing edge $e_{ij} = 1$. The components and edges in the template are 'virtual': the optimizer will retain only those necessary to satisfy the application constraints. A *library* $\mathcal{L}$ contains a set of 'real' components, which are assigned to the nodes of the template during the mapping process. Nodes and edges in the graph are labeled with different attributes, such as type, cost, delay, etc., corresponding to those from the components library $\mathcal{L}$.

Basis Constraints. A set of generic *basis* constraints, together with an objective function, such as cost minimization, define the topology selection and mapping problem. Topology selection consists of choosing the components from the template that are required to minimize the objective function. Mapping is the process of assigning a library component to the selected elements in the template. These constraints, which are automatically generated, formalize the exploration problem and ensure that the process behaves as expected. For instance, they impose that if a virtual component is used, then it is actually mapped to a library component. At the same time, it must be mapped to only one library component to avoid ambiguities.

Path and QoS Constraints. A network path, denoted as π , is a sequence of distinct nodes $v_0, \dots, v_n$. We use paths to define the interactions among the components. In our scenarios, a path connects an agent (a robot) to each of its required services, running on a server, through the wireless infrastructure. Formally, a path is associated with binary variables y_{ij}^π having the value 1 if the edge e_{ij} connects the node pair (v_i, v_j) within the path π , and 0 otherwise. A set of constraints is generated to let the optimizer decide how to generate a path consistently with the given template topology.

We implemented multiple QoS constraints to account for computation, communication, latency (E2E delay), and server capacity, denoted as α , β , λ and Φ . These constraints are expressed as linear integer constraints, ensuring compliance with a specific service maximum and minimum capacity requirements in terms of α and β . Additionally, a minimum λ is enforced to guarantee the timely execution of a service. The optimization involves determining the optimal values for two real variables $\alpha_{i,j,t}$ and $\beta_{i,j,t}$ allocated to server i and service j at any given time t as given by the following equations:

$$\alpha_{i,j}^{\min} \leq \alpha_{i,j,t} \leq \alpha_{i,j}^{\max}, \quad (6.1)$$

$$\beta_{i,j}^{\min} \leq \beta_{i,j,t} \leq \beta_{i,j}^{\max}, \quad (6.2)$$

which define the minimum and maximum available resources. The path constraints above determine the association between the server and the service. In addition, a constraint ensures compliance with the upper-bound threshold latency $\lambda_{i,j}^{\max}$ for each route that a robot is required to traverse (exceeding this threshold would lead to the robot failure in executing its real-time tasks):

$$\lambda_{i,j,t}^{\pi} \leq \lambda_{i,j}^{\max}. \quad (6.3)$$

Finally, a constraint ensures compliance with the requirement that the computation capacity allocated by a server to the services ($\phi_{i,j}$) at time t does not exceed the maximum computation capacity of the server ($\Phi^{\max}$). Exceeding the server capacity would lead to the robot failure in executing its real-time tasks.

$$\sum \phi_{i,j,t} \leq \Phi^{\max} \quad (6.4)$$

6.2.3 Automating the Simulation

Given an optimal architecture selected by ArchEx, all configuration files, simulation execution, and results are handled automatically. While the temporal dimension is modeled in *discrete space* during the MILP optimization phase, we use simulation to evaluate the solution in a *continuous space*, allowing us to observe the dynamic behavior of the architectural solution in terms of throughput and End-to-End (E2E) delay.

Having the topology information (coordinates of the APs and servers where the services are running) and the trajectory of the robots from the optimizer, a python script, denoted as ‘Simulation Manager’ in Figure 6.6, is used to manage the automatic code generation of the configuration files, while the interfaces to manage OMNeT++ simulations within the MinGW environment are implemented via batch scripts. This choice maintains the design flexibility and independence between the automation software and the simulator, allowing us to modify the simulator environment for further work with minimal effort if necessary. In the simulation manager, the first step is to clean up the old simulation files that have been generated to ensure reliable results. Secondly, starting from ArchEx outputs, all the files required by OMNeT++ (i.e., .ned for topol-

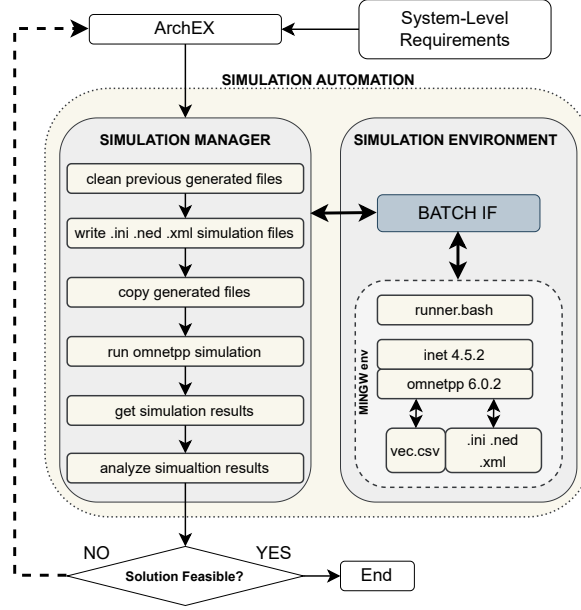


Figure 6.6: Framework Extension for Automated Simulation

ogy, .ini for parameter configuration, and .xml for robot trajectories) are automatically generated and moved within the MinGW environment. Afterwards, the simulation is started, and results are retrieved and analyzed.

The *runner.bash* script, located inside the simulation folder, is used to start and manage the simulation in the MinGW environment. It loads all the necessary .dll libraries and runs the simulation in command window mode (Cmdenv) to minimize the overhead associated with the graphical user interface (Qtenv).

After the simulation, the .anf results files are converted into .csv files, which are readable by *omnetppSimulationManager.py*, using the *opp_scavetool* OMNeT++ utility. The *RunSimulation.BAT* batch script serves as an interface between the Python code and the bash runner, ensuring that the simulation process completes before proceeding.

Finally, *omnetppSimulationManager.py* retrieves throughput and End-to-End (E2E) delay by parsing the .csv files.

6.2.4 Evaluation and Results

We begin our model evaluation by considering two scenarios in an indoor robotic environment. Our objective is to determine a feasible and optimal path through the access points (APs) for the robots to reach the services at each time step.

In the first scenario, we consider one robot, five access points, and two services. In

the second scenario, we have two robots competing for resources. The latency bound is set to 1 ms.

After successfully executing the optimization problem, the architecture (Figure 6.7) and all necessary files are generated as output. These outputs illustrate the node selection and the paths orchestrated by ArchEx.

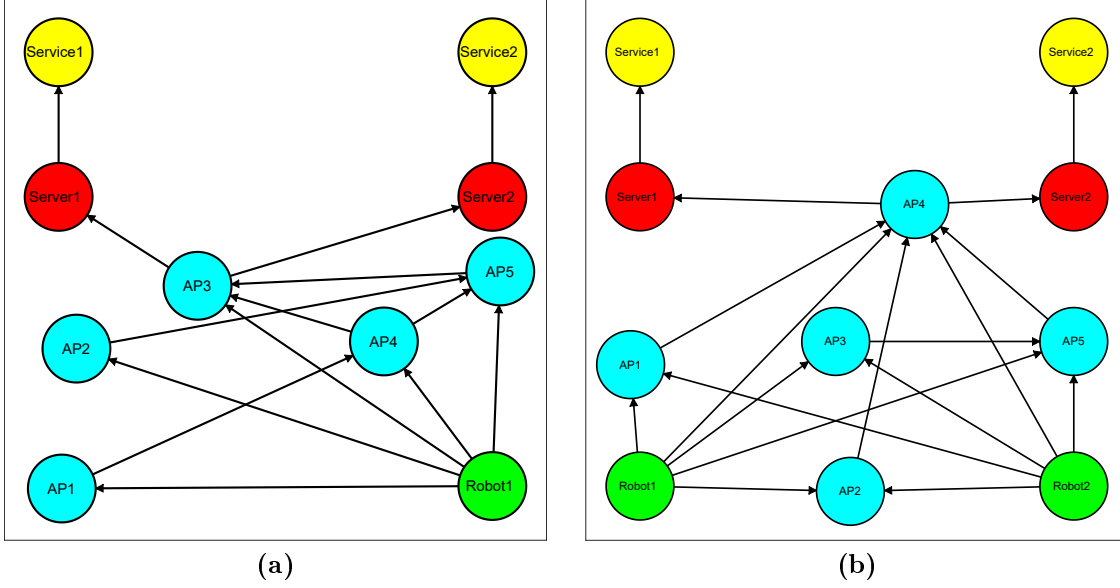


Figure 6.7: Generated Architectures from ArchEx, for One Robot Scenario (a) and Two Robots Scenario (b).

For example, in the first scenario, the links between AP1 and AP2 are never used. Similarly, in the second scenario, the links between AP1 and AP5, and between AP2 and AP3, are never used by the robots to reach a service.

The Table 6.5 and Table 6.6 reports the trajectories of the robots and the paths they take to reach a service hosted on a server at each time step.

In these tables, it is easier to see which paths the robot (or robots) use to reach a particular service at each time step. The selected topology specifies how the network infrastructure should be designed (e.g., connections between APs), based on the robot trajectories. In other output files, all the paths are stored instead. In both the first and second scenarios, the optimization shows that robot1 always reaches service 1. In the second scenario, the new robot always reaches service 2.

After each optimization phase, we perform simulations for the two network architectures using OMNeT++ 6.0.2 with the INET 4.5.0 extension. The simulation scenarios are modeled with a mixed wired and wireless network, featuring moving robots, APs,

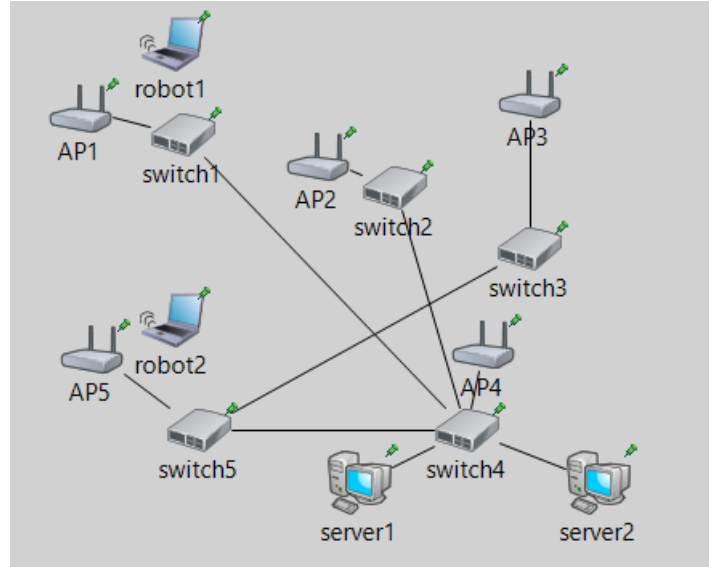
Table 6.5: Robot Trajectories and Paths in First Scenario

Robot 1				
Time	x [m]	y [m]	Service	Path
t1	31.6	41.2	1	AP1, AP4, AP3, SERVER1
t2	38.9	113.9	1	AP2, AP5, AP3, SERVER1
t3	720.9	21.1	1	AP3, SERVER1
t4	657.7	415.9	1	AP4, AP5, AP3, SERVER1
t5	27.0	427.6	1	AP5, AP3, SERVER1

Table 6.6: Robot Trajectories and Paths in Second Scenario

Robot 1					Robot 2			
Time	x [m]	y [m]	Service	Path	x [m]	y [m]	Service	Path
t1	31.6	41.2	1	AP1, AP4, SERVER1	31.0	437.6	2	AP5, AP4, SERVER2
t2	38.9	113.9	1	AP2, AP4, SERVER1	685.7	435.9	2	AP4, SERVER2
t3	720.9	21.1	1	AP3, AP5, AP4, SERVER1	740.9	25.1	2	AP3, AP5, AP4, SERVER2
t4	657.7	415.9	1	AP4, SERVER1	340.0	123.0	2	AP2, AP4, SERVER2
t5	27.0	427.6	1	AP5, AP4, SERVER1	51.6	41.2	2	AP1, AP4, SERVER2

and servers. An example of the resulting architecture in the simulator GUI for the second scenario is shown in Figure 6.8).

**Figure 6.8:** Second Scenario from OMNeT++ Graphic User Interface (Qtenv)

Switches are used to facilitate communication between multiple APs and to model multi-hop transmission.

Each robot transmits radar data at 5.2 Mbps to the server hosting the requested service. All simulations are executed on a ZBOOK with 32 GB of RAM and an Intel Core i7-11850H processor. We simulate the first 80 seconds of operation for each scenario, which take 46.05 seconds and 87.75 seconds, respectively, to complete. Note that in the second scenario, nearly double the time is required for the simulation to complete due to the increased data transmission.

The simulation results are reported in Table 6.7.

Table 6.7: Throughput and E2E Delay for Indoor Robotics Scenario

Scen.	Robot	Min E2E	Max E2E	Mean E2E	Mean Throughput
1	Robot1	0.65 ms	0.94 ms	0.82 ms	5.02 Mbps
2	Robot1	0.65 ms	1.70 ms	0.85 ms	5.11 Mbps
2	Robot2	0.66 ms	1.48 ms	0.83 ms	4.99 Mbps

Throughput is less than 5.2 Mbps because data is not transmitted during the authentication steps required to associate each robot with a new AP when a handover between two cells takes place, as can be seen also from Figure 6.9. The required upper limit of 1 ms is met for the first scenario and the average measured throughput is about 5 Mbps (Figure 6.10).

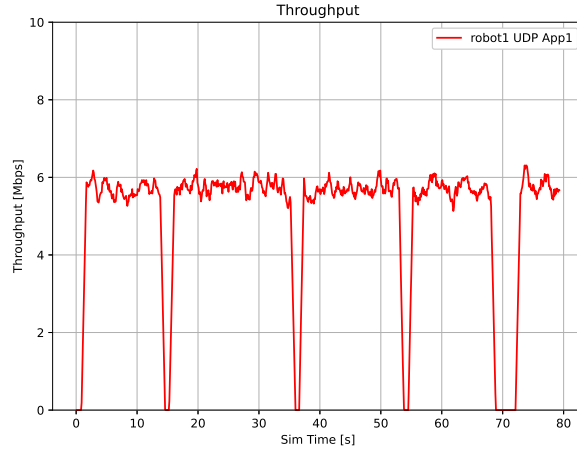


Figure 6.9: Throughput in Scenario 1 With One Robot

For the second scenario, the average throughput is once again around 5 Mbps, and the average upper limit of 1 ms for E2E delay is met for both robots. However, the maximum latency does not satisfy the constraints. Figure 6.11 shows the throughput for both robots, while Figure 6.12 shows the E2E delay for both robots.

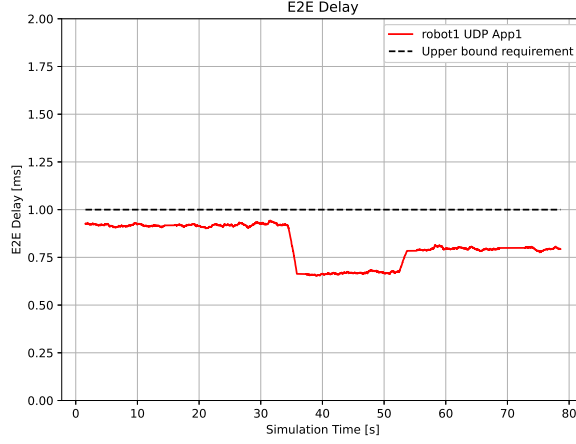


Figure 6.10: E2E Delay in Scenario 1 With One Robot

While throughput remains relatively constant (as the AP can handle the aggregate throughput of the two robots), the E2E delay increases when a robot tries to reach a server via a longer path that traverses several access points using a multi-hop protocol. Additionally, when the two robots transmit within the same cell to the same AP, the E2E delay increases to a maximum of 1.7 ms (for Robot1), which could result in errors for robots performing real-time tasks.

This behavior, which was not modeled in the original problem formulation, can be fed back to the optimizer as extra constraints to enforce the use of alternative paths or the instantiation of additional APs. If the E2E delay requirements are not met during same-cell transmission, we have identified a dynamic behavior from the simulation that cannot be predicted by the optimizer. This was one of our goals, as it allows us to improve the model and converge toward a stable solution.

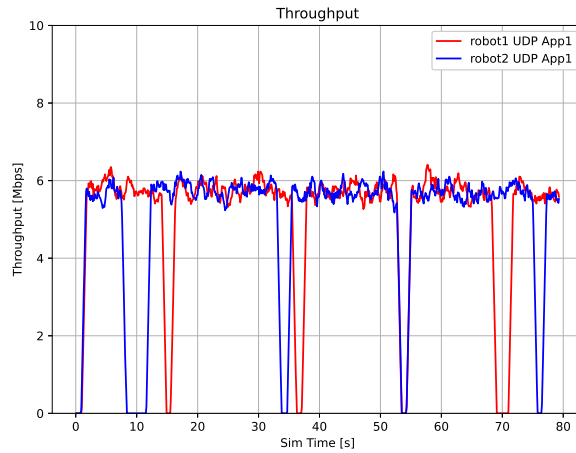


Figure 6.11: Throughput in Scenario 2 With Two Robots

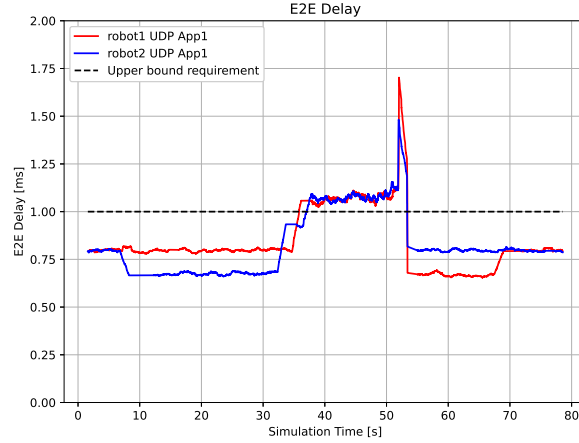


Figure 6.12: E2E Delay in Scenario 2 With Two Robots

A third scenario uses a more complex arrangement with 15 APs, each with a corresponding server. Two robots move around the space, connecting to three services (two for robot 1, and one for robot 2).

The floor plan is shown in Figure 6.13 together with the robot trajectories, and the location of the APs and the servers in the template. The figure also shows some representative communication paths.

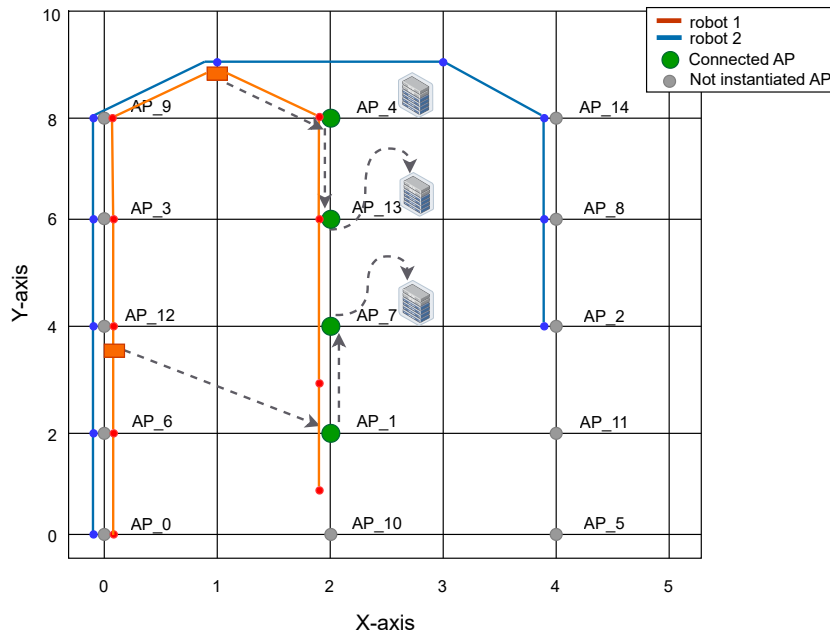


Figure 6.13: Third Scenario Floor Plan and Trajectories

Each robot sends compressed camera data at 24 Mbps, which corresponds to the maximum link capacity. In this case, we do not define specific bounds on latency and throughput. The optimizer selects 4 out of the 15 APs, located in the middle section of the deployment, which guarantee complete coverage of the area, while three of these APs also connect to the corresponding servers to run the services. Specific communication paths and robot trajectories selected by the optimizer are shown in Table 6.8.

Table 6.8: Robot Trajectories and Paths in Third Scenario

Robot 1					Robot 2				
Time	x [m]	y [m]	Service	Path	x [m]	y [m]	Service	Path	
t0	0	0	0 (S7), 1 (S13)	(AP1, AP7, S7), (AP1, AP7, AP13, S13)	0	0	10 (S4)	AP1, AP7, AP13, AP4, S4	
t1	0	200	0 (S7), 1 (S13)	(AP1, AP7, S7), (AP1, AP7, AP13, S13)	0	200	10 (S4)	AP1, AP7, AP13, AP4, S4	
t2	0	400	0 (S7), 1 (S13)	(AP1, AP7, S7), (AP1, AP7, AP13, S13)	0	400	10 (S4)	AP1, AP7, AP13, AP4, S4	
t3	0	600	0 (S7), 1 (S13)	(AP4, AP13, AP7, S7), (AP4, AP13, S13)	0	600	10 (S4)	AP4, S4	
t4	0	800	0 (S7), 1 (S13)	(AP4, AP13, AP7, S7), (AP4, AP13, S13)	0	800	10 (S4)	AP4, S4	
t5	100	900	0 (S7), 1 (S13)	(AP4, AP13, AP7, S7), (AP4, AP13, S13)	100	900	10 (S4)	AP4, S4	
t6	200	800	0 (S7), 1 (S13)	(AP4, AP13, AP7, S7), (AP4, AP13, S13)	300	900	10 (S4)	AP4, S4	
t7	200	600	0 (S7), 1 (S13)	(AP4, AP13, AP7, S7), (AP4, AP13, S13)	400	800	10 (S4)	AP4, S4	
t8	200	300	0 (S7), 1 (S13)	(AP1, AP7, S7), (AP1, AP7, AP13, S13)	400	600	10 (S4)	AP4, S4	
t9	200	100	0 (S7), 1 (S13)	(AP1, AP7, S7), (AP1, AP7, AP13, S13)	400	400	10 (S4)	AP1, AP7, AP13, AP4, S4	

In particular, robot 1 uses service 0 hosted on server 7 and service 1 hosted on server 13, while robot 2 uses service 10 hosted on server 4. While we do not utilize all its features, the optimizer employs a wireless propagation and channel model that could account for obstacles (not modeled) to determine the visibility between the robot agents and the access points based on their relative positions.

Figure 6.14 and Figure 6.15 show the results of the simulation.

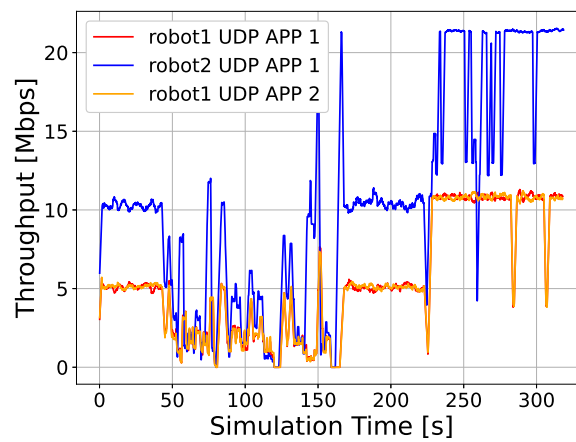


Figure 6.14: Throughput in Scenario 3

We observe how the overall achieved communication throughput is significantly degraded at the beginning of the simulation, when the robots move close to each other

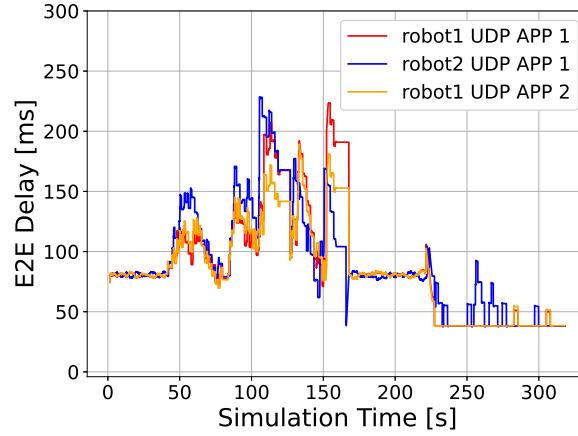


Figure 6.15: E2E Delay in Scenario 3

and share the same access point. While the optimizer does employ a channel model to determine which APs are reachable at every time step and construct the paths [43], the simulator can more precisely account for both collisions and transmission errors, which are more frequent at the borders between cells, just prior to the handovers. This is clearly visible in the 50–150 s time range, where the robots interfere with each other and several APs are in view. A more precise scheduling of the communication could potentially address some of these issues. In addition, the simulation shows that robot 1 experiences lower throughput, since the resources are shared by the two services it uses. The E2E delay presents a similar, but opposite, evolution, and points to the same critical sections. Note that throughput and latency need not be inverse of each other. In particular, robot 1 experiences latencies similar to robot 2 despite using two separate services.

The simulation for this scenario runs for 776 seconds, highlighting the much higher complexity of performance evaluation due to the increased rate of data transfer and longer simulated time. This underscores the importance of combining the optimizer and the simulator. The optimizer selects only the most promising architectures, guiding a potentially expensive performance evaluation to much fewer cases. Conversely, simulation leads the optimizer to exclude solutions that turn out to be infeasible, reducing the search space. The effect is, therefore, a conceivably much faster convergence toward a solution that satisfies the application requirements.

6.2.5 Discussion

In this second evaluation, we integrated symbolic optimization-based design space exploration with event-driven simulation. This technique combines abstract architectural models amenable to MILP optimization algorithms with performance and behavior models that reveal the detailed behavior of the system. The idea is to use the optimizer to find feasible candidate architectures, which are then evaluated by the simulator within a loop that converges to an optimal solution by feeding back information in the form of constraints to satisfy the network QoS requirements.

In the next section, we will complete our framework evaluation by finalizing our methodology with the automated iterative simulation-in-the-loop (SIL) strategy, logically represented by the dotted line in Figure 6.6.

We will demonstrate how the bandwidth requirements will be automatically adjusted to reach the final system solution iteratively.

We apply the framework to the E/E architecture for off-road vehicles (the key objective domain of this dissertation) and evaluate the framework through larger case studies.

6.3 Combining Optimization and Simulation for ORVs Architectural Design

This final evaluation aims to finalize the development and validation of our design space exploration framework. The design process is fully automated, combining optimization and simulation, iteratively regenerating constraints to converge to the final design solution.

A comprehensive representation that underscores the entirety of the framework associated with the methodology is illustrated in Figure 6.16.

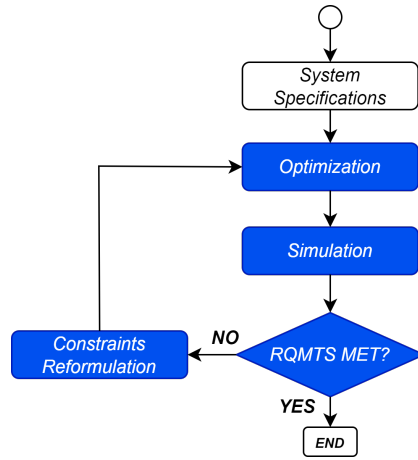


Figure 6.16: Level of Development of the Framework in the Current Evaluation: The optimization, simulation and constraint reformulation process is fully automated

We conducted two optimizations: one to find the bandwidth limit at the backbone, determining the maximum bandwidth constraint that leads to feasible architectural solutions, and another to obtain an architectural solution that minimizes system power consumption.

We then performed various experiments to analyze the latency and jitter of the obtained architecture. Additionally, we identified the best design parameters for the CAN-to-Ethernet gateway, examining the effects of changing the hold-up factor (time before releasing aggregated packets of data from the CAN domain to the Ethernet domain) on the latency and jitter of the entire stream, discussing the trade-offs. We derived Pareto fronts to visualize the trade-offs between cost, weight, and power consumption.

Finally, we conducted a qualitative analysis to test the scalability of our framework in terms of design time when increasing the number of sensors or the capacity of the backbone link.

This evaluation focuses on the design optimization of off-road vehicle E/E (Electrical/Electronic) architecture. As the application domain has already been discussed in the introduction, we will not provide an overview here. In the following sections, we will detail the problem formulation, briefly revisiting the process involved, and then thoroughly presenting the evaluation setups and results.

6.3.1 Problem Formulation

As previously discussed, we evaluate our framework by performing a Design Space Exploration (DSE) for a zone-based off-road automotive E/E (Electrical and Electronic) architecture. The process begins with the selection of hardware components necessary for implementing assisted and autonomous driving functions, choosing from a set of RADARs and/or cameras.

The optimizer output includes both the selection of 'virtual components' (components instantiated at the architectural level, represented by decision variables set to one in the adjacency matrix) and 'real components' (commercially available hardware selected for system implementation, represented by decision variables set to one in the mapping matrix).

The library of commercial sensors comprises RADAR and automotive cameras with varying characteristics such as data rate, cost, power consumption, and weight. All physical layers are 100BASE-T1, supporting speeds of 100 Mbps, 1 Gbps, and 10 Gbps within 15 meters.

Starting from an oversized template and a component library, the optimizer takes system requirements (bandwidth, cost, weight, and power consumption) and an objective function as inputs to generate a candidate solution.

Post-optimization, the simulation and result extraction are performed automatically to verify that system requirements related to latency and load on both the Ethernet backbone and the CAN network are met. If the solution is not feasible (e.g., excessive latencies, network load, or scheduling issues at node initialization), the constraints for the optimizer (e.g., maximum allocable bandwidth percentage) are reformulated. This iterative process continues until it converges to a feasible solution or one that meets a specific goal.

6.3.2 Architectural Template and CAN Gateway Design

In this problem, the existing CAN architecture is fixed due to industrial design constraints. It is not feasible to redesign the entire network architecture, as the current

vehicular functions (throttling, braking, etc.) must remain unchanged. Instead, new functionalities must be integrated into the existing vehicular platform. The high-speed architecture (Ethernet), which supports new functionalities related to assisted and autonomous driving, is selected by the optimizer.

A conceptual diagram illustrating the integration of the new system, which implements additional functionalities within the existing vehicular network composed of a CAN bus, is shown in Figure 6.17.

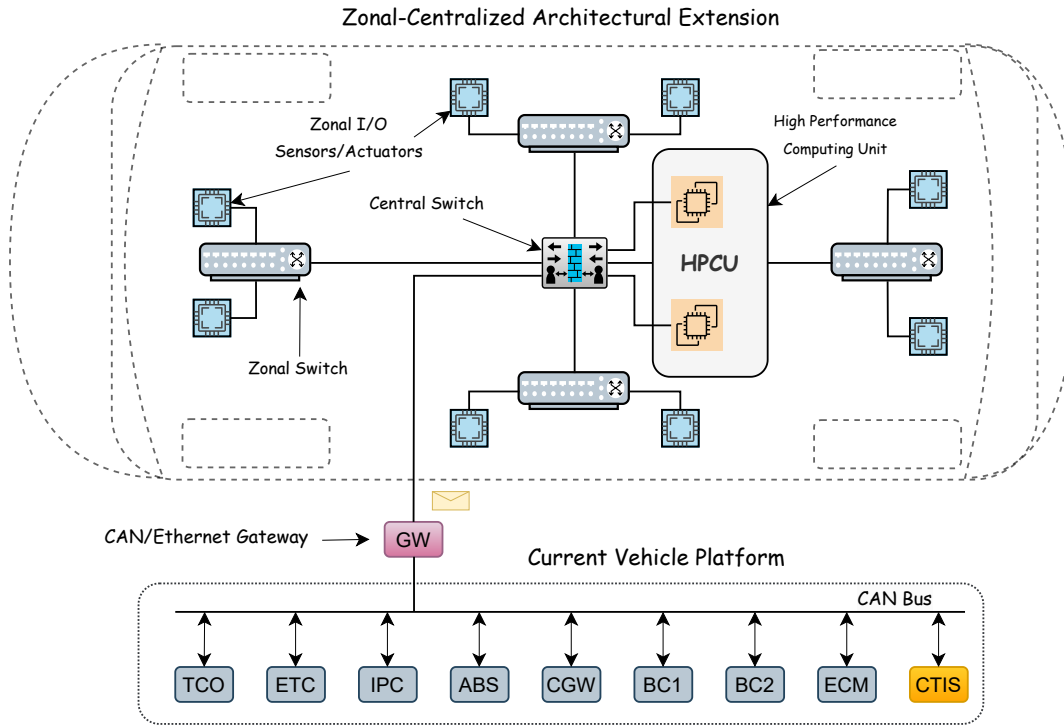


Figure 6.17: Architecture Conceptual Scheme.

In our design, we use Class A AVB streams (guaranteed maximum latency of 2 ms) for high data rate sensor transmissions that perform assisted driving tasks, and TTEthernet (deterministic constant latency) to control two DBW modules. This allows us to test the network ability to support different communication standards and observe how their coexistence affects network communication.

The template is a zonal-centralized architecture, consisting of a central ECU (HPCU, also called ADCU) responsible for executing logic related to assisted and/or autonomous driving functions (obstacle and pedestrian detection, traffic sign recognition, driver monitoring, and speed assist). A set of network switches forwards data from sensors

located in various areas of the vehicle to the central ECU, as well as control data from the central ECU to the SBW modules that provide steering control.

The switches also forward data converted from CAN to Ethernet, which includes various vehicle data (vehicle speed, engine RPM, steering angle, current gear, power-take-off state, transfer case status, etc.) necessary for implementing various ADAS functionalities. Therefore, a significant and mandatory step in this work is the design and definition of the gateway implementation specification.

Starting from the homologation requirements (GSR), we derived all the J1939 messages (Table 6.9) that must be routed from the CAN domain to the Ethernet domain.

Table 6.9: Gateway Forward Table: J1939 CAN messages (information and transmission rates) which need to be routed to the Ethernet domain to meet GSR requirements (SAS and TCO are the steering angle sensor and tacograph ECUs, respectively).

Node/Function	GSR Requirement	J1939 Msg Name	Information	Tx Rate [ms]
ABS	DDAW, REIS	EBC1	Brake Switch Information	20
ABS	AIF,BSIS, DDAW, ISA, MOIS, REIS	EBC2	Vehicle Speed	100
ABS	DDAW	VDC2	Yaw rate	100
ABS	AIF,BSIS, DDAW, ISA, MOIS, REIS	HRW	Vehicle Speed	20
TCO	AIF, BSIS, DDAW, ISA, MOIS, REIS	TCO1	Vehicle Speed	20
TCO	AIF, BSIS, DDAW, MOIS, REIS	VDHR	OdometerData	1000
TCO	AIF, BSIS, DDAW, MOIS, REIS	TD	Time and Date	1000
ECM	AIF, DDAW, MOIS, BSIS, REIS	CCVS1	Clutch Status, Parking Brake status	100
ECM	AIF, DDAW, REIS	EEC2	Throttle Switch Status	100
ECM	AIF, DDAW, ISA, REIS	EEC1	Engine Speed	10
ETC	DDAW, BSIS, MOIS, REIS	ETC5	Rear gear Information	1000
ETC	DDAW, REIS	PTO	Power Take-Off information	100
ETC	DDAW, REIS	ETC1	Power Take-Off information	10
ETC	DDAW, BSIS, ISA, MOIS	ETC2	Gear Information	100
BCM	BSIS, DDAW, MOIS, REIS	LD	Lighting Data	50
BCM	DDAW	EAC1	Differential Locks State	500
BCM	BSIS, DDAW, MOIS, REIS	TCI	Transfer Case State	1000
SAS	DDAW	SAS	Steering Wheel Angle	10

This step is critical for designing the gateway and understanding, through simulation, the impact of CAN routing on the Ethernet network load and the latencies of various streams. These dynamic cross-traffic behaviors (including CBS and message scheduling for different standards) in the vehicular network are challenging to model within the optimizer, so we rely on simulation for their analysis.

The switches in the network must support both AVB and TTEthernet communication standards. To achieve this, we adopt the Transmission Selection Algorithm for AVB and time-triggered traffic proposed by Rumpf et al. [77], which is already implemented in the simulated switch module (6.18).

In their algorithm, time-triggered traffic is not allowed to be delayed by any other frame, thus TT frames must have the highest priority. This requirement conflicts with the service class A of IEEE 802.1Qav, which uses the highest priority for its end-to-end performance guarantees. Therefore, changes were made to the queuing and

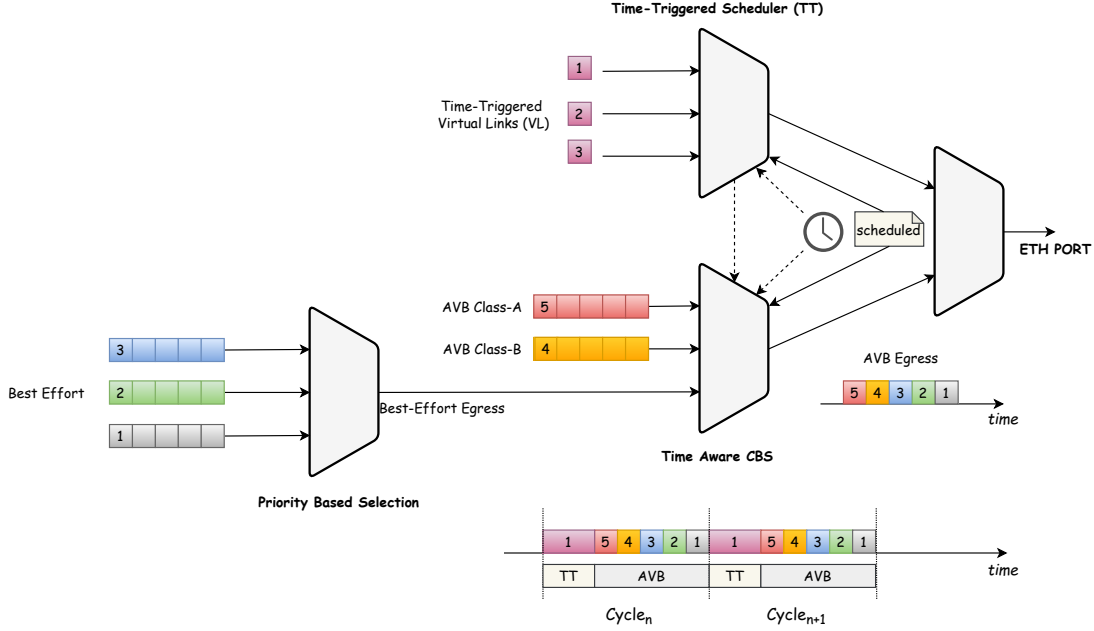


Figure 6.18: Transmission Selection Algorithm for AVB and Time-Triggered Traffic: CBS uses clock and schedule, enabling TT communication with no interference

scheduling strategy of the AVB frames. The time-triggered send-windows and the domain-wide synchronized clock are used as inputs for the Credit Based Shaper (CBS) algorithm, which performs the required controls. Consequently, this new time-aware CBS introduces further delays for all traffic classes of AVB. They noted that the latency guarantee of SR class A (2 ms over 7 hops) is twice as long (4 ms) as in the original standard. However, in our work, this does not significantly affect the transmission and is negligible, as the latency is much lower than 2 ms.

Using these techniques, we can support multiple protocols on the network. AVB is used to gather data from radars and cameras, while TTEthernet manages data control traffic to actuators, which is critical and requires a deterministic delay. This is achieved by reserving a high-priority transmission/reception time slot for TT frames, even if it slightly delays the AVB streams.

An example of how the system architecture is represented within the simulation environment is illustrated in Figure 6.19.

The system architecture, as depicted, includes sensors, ECUs, and switches instantiated by the optimizer across various vehicle zones. Additionally, it showcases the ECUs within the CAN Bus network (representing the pre-existing design), the gateway bridging the Ethernet and CAN domains, and the two SBW modules utilized for

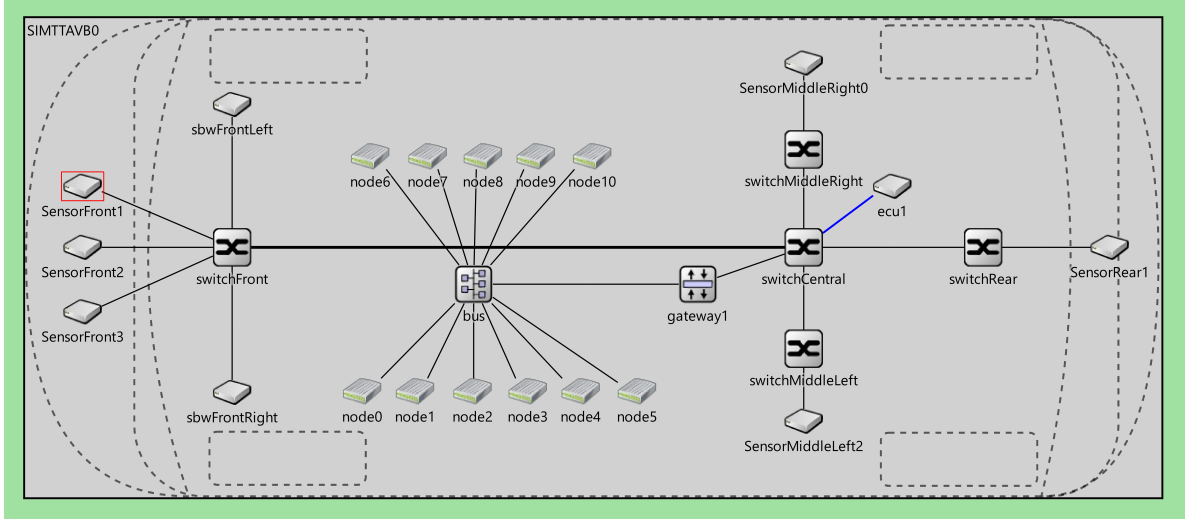


Figure 6.19: CoRE-RG Framework: Simulated System Architecture

steering.

Finally, we would like to mention that, as in the initial work, we have integrated an ECU called CTIS into the CAN network. This ECU performs electronically controlled inflation/deflation based on the type of terrain (e.g., sand, track) and monitors tire pressure. Therefore, when analyzing the CAN bus load, it is essential to verify that the new messages required to implement GSR functions (i.e., warnings to the driver in specific situations) and the messages introduced by this new ECU do not increase the bus load beyond the threshold discussed in the next section.

6.3.3 System Requirements and Evaluation Setup

Regarding system requirements, the total load of the CAN bus, including new ADAS messages that must be captured from the instrument cluster to alert the driver, must not exceed 50% to ensure that all CAN messages are sent according to their transmission period, minimizing the risk of collisions within the network [47].

Additionally, the maximum latency of each sensor/control stream must be less than 2 ms to ensure that perception tasks are completed within the required time limit (strict latency constraint upper bound defined for Class D SAE classes [32]). Therefore, we set this as the maximum latency limit for streams traveling over Ethernet using Class-A AVB streams.

The control data for the SBW modules must have jitter as close to zero as possible to ensure reliable communication [53], as it is a time-critical task. Consequently, we use TTEthernet scheduled traffic to transmit these control commands to the SBW actuators

and verify if it meets the requirement.

All evaluations are run on a ZBOOK with 32 GB of RAM and an Intel Core i7-11850H. Our initial design process executes each simulation for 5 s of simulated time, as this duration is significantly greater than 0.5 s, which is sufficient to determine if scheduling through the SRP protocol for all streams is feasible.

6.3.4 System Bandwidth Optimization

In the first evaluation of the tool, our goal is to determine the maximum bandwidth constraint for the backbone to obtain a feasible solution where all streams and messages can be scheduled and transmitted. Initially, we set 75% as the maximum bandwidth limit for non-cross traffic, as specified by AVB standards. However, the simulator indicates that this solution is unfeasible because the dynamic aspects of the network and electronic integration, such as programming and forwarding CAN messages to Ethernet, are not modeled by the optimizer. Therefore, we must lower the maximum bandwidth constraint. To find the maximum bandwidth constraint for which a feasible solution exists, several approaches can be adopted, as already discussed in Section 4.4. A brute-force approach is impractical due to the simulation time required (5 s of virtual simulation requires about 2800 s of real-time simulation). The constraint can be instead modeled as an increasing unimodal cost function, requiring the selection of a suitable optimization algorithm, as an inappropriate choice may perform no better than random search [107]. Hill-climbing and Newton’s methods are unsuitable due to tuning complexity and the absence of second derivatives, respectively. Stochastic methods, such as simulated annealing and genetic algorithms, are effective for complex spaces but introduce randomness that slows convergence. Gradient-based methods, while challenged by non-differentiability at the peak, can handle this issue using adaptive learning rates. The golden-section search, although not ideal due to requiring two function evaluations per iteration, remains a viable option for unimodal functions when computational cost is managed carefully. Consequently, the gradient-ascent algorithm (Algorithm 1) and the golden-section search (Algorithm 2) were selected as candidates. To evaluate these two methods, we conducted an experiment to determine the number of simulations required for convergence. The cost function, as previously described, has a single maximum, beyond which it drops to zero (unfeasible solution). Depending on the scenario (e.g., design optimization goals, constraints, and library components), the maximum bandwidth constraint which ensures a feasible solution is unknown. As a result, several simulations are necessary to identify this value. To accomplish the selection of

the most appropriate method, we used a mathematical model to represent the cost function. Then, the two algorithms were applied to this model, varying the bandwidth peak between 50% and 90%. Considering that the golden-section search requires two simulations per iteration, the evaluation results are presented in Figure 6.20.

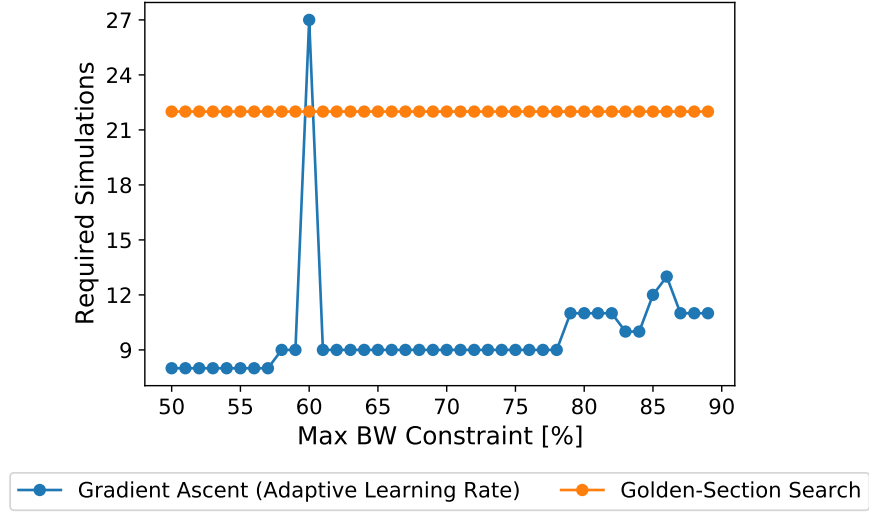


Figure 6.20: Number of Simulations Required for Gradient-Ascent and Golden-Section Search Across Various Scenarios (Maximum Bandwidth Constraints Ranging from 50% to 90%)

The comparison is performed to ensure that each algorithm converges with a maximum error of 0.5%. As shown, the gradient ascent algorithm with an adaptive learning rate outperforms the golden-section search, except when the maximum bandwidth constraint is 61%. The key advantage of the golden-section search lies in its determinism, guaranteeing convergence to the solution every time with a consistent number of simulations. In contrast, the gradient ascent method with an adaptive learning rate is less deterministic but achieves faster convergence, requiring fewer iterations. To complete the analysis, we also reported how the maximum bandwidth constraint changes at each iteration. This was done for scenarios where the peak corresponds to 63% of the bandwidth (i.e., 63 Mbps) as shown in Figure 6.21, and 85% of the bandwidth (i.e., 85 Mbps) as illustrated in Figure 6.22.

As observed, the dashed red line marks the peak beyond which the MILP optimizer yields infeasible solutions. The light red area highlights maximum bandwidth constraint values that lead to infeasibility. Both algorithms iteratively adjust the constraint value, converging to the maximum with a guaranteed error of 0.5%. It is important to note that, although the golden-section search requires 11 iterations, it necessitates two sim-

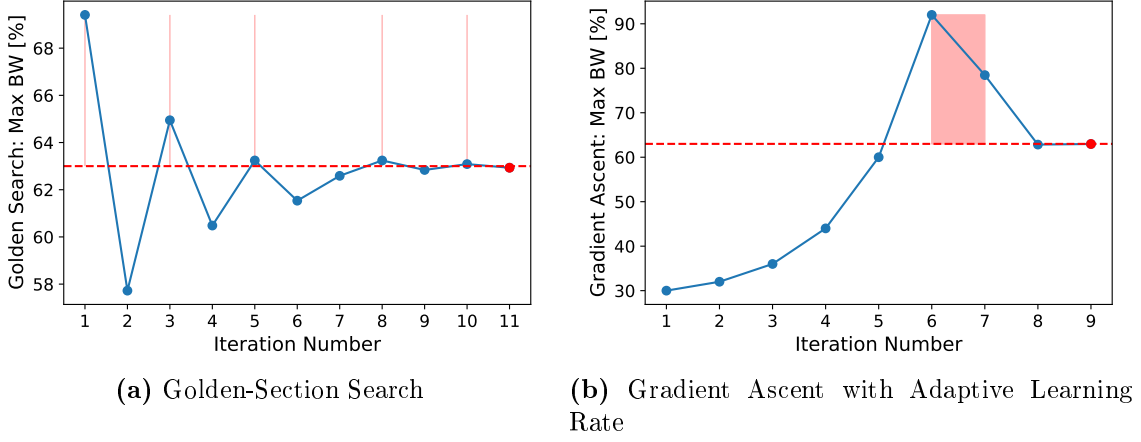


Figure 6.21: Maximum bandwidth constraint update during iterations, with the gradient ascent with adaptive learning rate and the golden-section search (peak at 63%)

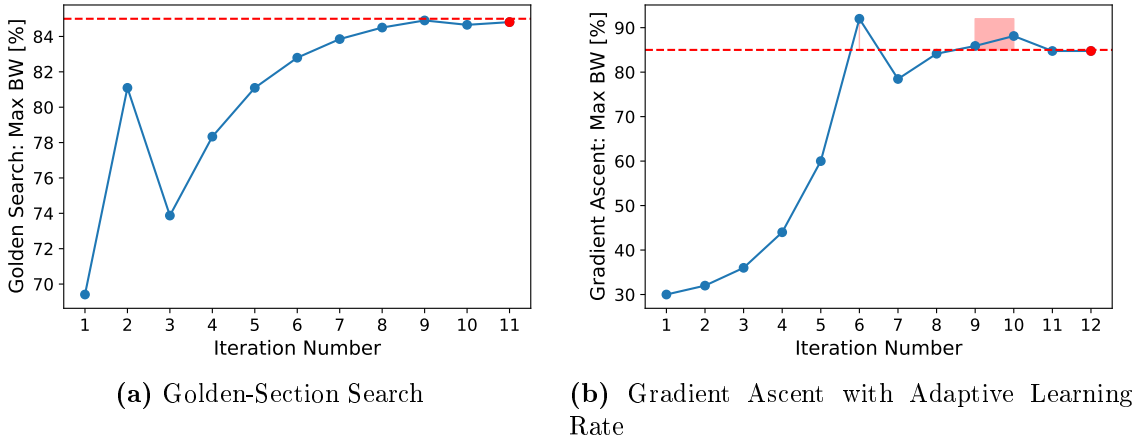


Figure 6.22: Maximum bandwidth constraint update during iterations, with the gradient ascent with adaptive learning rate and the golden-section search (peak at 85%)

ulations per iteration to reach the final solution, making it less practical in terms of simulation cost.

Based on this analysis, we opted to use the gradient ascent algorithm with an adaptive learning rate to achieve our objective. Consequently, we proceeded to determine the maximum bandwidth constraint for the backbone using our framework, validating the results through real simulations. For this and subsequent evaluations, we employed a backbone connection with a capability of 100 Mbps.

Our design solution is found in 9 iterations and 25552 s (approximately 7 hours), with most of the time attributed to the simulation. This underscores the importance of combining the optimizer and the simulator. The optimizer selects only the most

promising architectures, guiding a potentially expensive performance evaluation to significantly fewer cases. Conversely, the simulation helps the optimizer exclude solutions that turn out to be infeasible, thereby reducing the search space. The result is a much faster convergence toward a solution that satisfies the application requirements. The resulting first architecture and the implementation are shown in Figure 6.23, which shows the selected topology, and Figure 6.24, providing the correspondence between virtual and real library components.

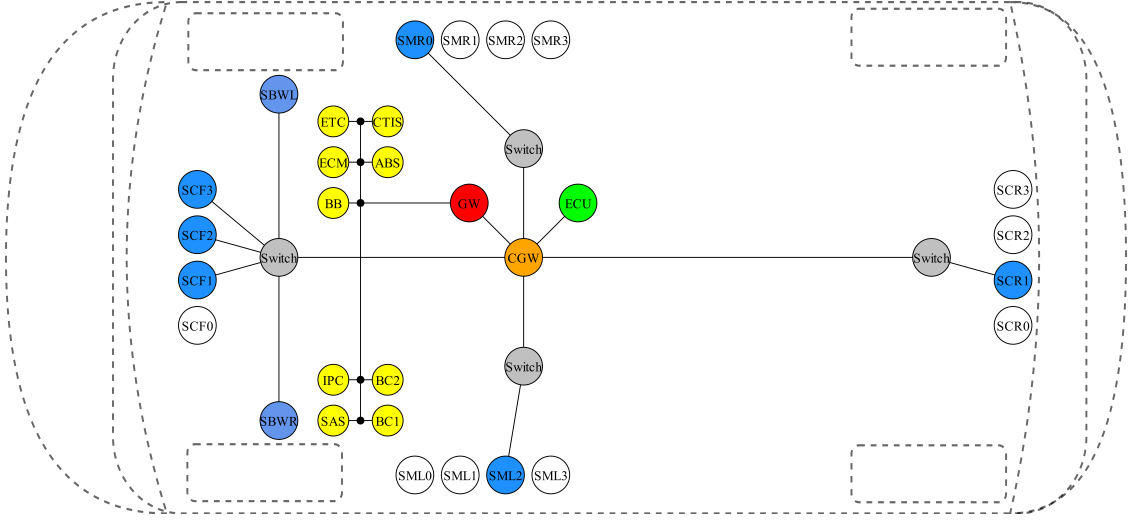


Figure 6.23: Optimized E/E Architecture for Maximized BW Constraint (Colored Components are Those Selected by the Optimizer)

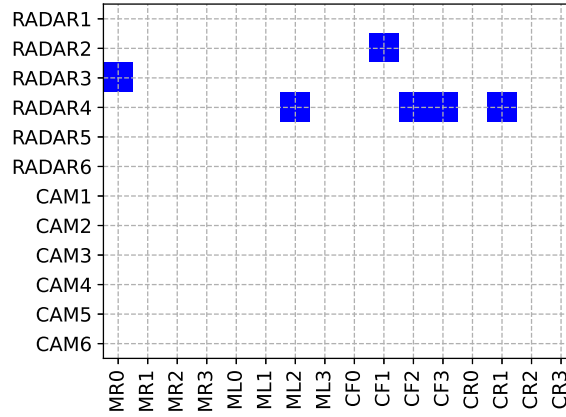


Figure 6.24: Optimized Mapping for Maximized BW Constraint (Only Sensor Library Shown)

The results show that 6 RADARs are selected. Extended simulation data (we launched a more time-consuming simulation only after finding the optimal solution)

regarding latency and jitter are reported in Table 6.10. From the latency plots (Figure 6.25), it is possible to observe, as in previous work, the effect of the online Credit-Based Shaping (CBS) mechanism used in AVB for online bandwidth allocation.

The Credit parameter for each stream class decreases during transmission (send-Slope) and increases otherwise (idle-Slope). When the credit associated with each SR queue is non-negative, pending frames can be transmitted. The latency of TTEthernet deterministic traffic is fixed as expected.

In our implementation, the central ECU sends two time-triggered control frames for the SBW modules every 2 ms, one controlling the right wheel and the other the left wheel, each using a bandwidth of 50 kbps (100 kbps total). The deterministic latency is guaranteed by predefined slots for the transmission of TT Frames. This type of traffic satisfies both the latency requirement bound of 2 ms and the jitter requirement, as both the minimum and maximum jitter are very small (less than 1 μ s), indicating stable and continuous transmission without significant fluctuations that could otherwise cause unwanted delays in command transmission.

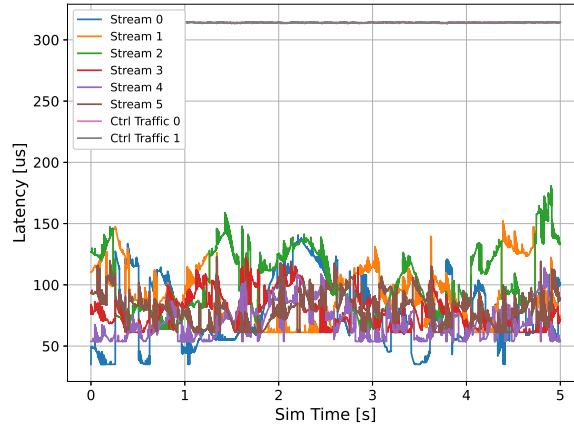


Figure 6.25: Stream Latencies of the Optimized E/E Architecture for Maximized BW Constraint

The introduction of additional messages to satisfy GSR requirements (information to warn the driver under certain conditions) and the addition of an ECU to perform inflation/deflation and monitoring of tire pressure (to optimize driving in off-road conditions) results in a CAN bus load increase from 32.03% to 48.02%. This remains within the required 50% bound. The measured bandwidth occupation in the Ethernet backbone is 63.43%. Finally, the total system cost is 6150 EU.

To summarize, the network load (both in the CAN and Ethernet domains), latency and jitter requirements, as well as other design criteria such as weight, power con-

Table 6.10: Extended Simulation Results of the Optimized E/E Architecture for Maximized BW Constraint

Traffic	Src	Dst	BW [Mbps]	Protocol	Max Latency [μ s]	Mean latency [μ s]	Max Jitter [μ s]	Mean Jitter [μ s]
Stream 0	CF1	ECU1	5	AVB Class-A	140.75	76.77	92.31	0.41
Stream 1	CF2	ECU1	12	AVB Class-A	152.25	89.23	62.83	0.51
Stream 2	CF3	ECU1	12	AVB Class-A	180.87	100.79	51.91	0.46
Stream 3	ML2	ECU1	12	AVB Class-A	122.53	77.09	46.25	0.48
Stream 4	MR0	ECU1	10	AVB Class-A	113.96	68.16	46.20	0.48
Stream 5	CR1	ECU1	12	AVB Class-A	123.07	78.97	47.67	0.48
Ctrl Traffic 0	ECU1	SBWR	0.05	TTEthernet	315.11	314.06	0.93	0.24
Ctrl Traffic 1	ECU1	SBWL	0.05	TTEthernet	314.93	314.06	1.03	0.25

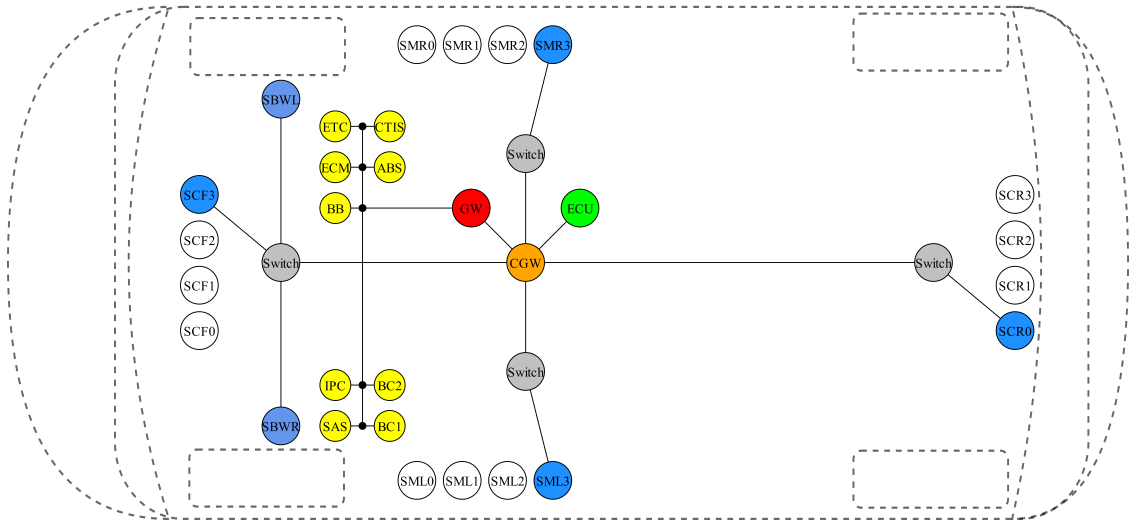
sumption, and cost, are all satisfied. This indicates that the entire design exploration framework functions as expected.

In the next section, we conduct another experimental evaluation. We use the bandwidth constraint obtained in this design step as an input for the optimizer to find an architectural solution that minimizes system power consumption.

6.3.5 System Power Consumption Optimization

As anticipated, in the second evaluation of the tool, we aim to minimize the power consumption of the entire system. In this case, a feasible solution is found in just one iteration (approximately 45 minutes), highlighting how resolving the previous bandwidth constraint issue facilitates pursuing other project objectives. The obtained system cost is 5750 EU, while the system electrical power consumption is 17.3 W.

The design results are reported in Figure 6.26, Figure 6.27 and Table 6.11.

**Figure 6.26:** Optimized E/E Architecture for Minimized Power Consumption (Colored Components are Those Selected by the Optimizer)

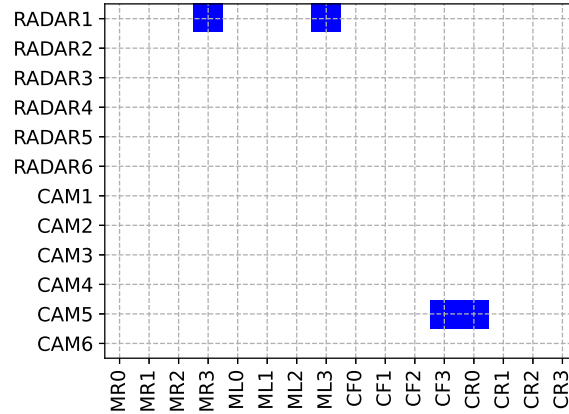


Figure 6.27: Optimized Mapping for Minimized Power Consumption (Only Sensor Library Shown)

In this case, the optimizer selects 2 cameras and 2 RADARs. This outcome aligns with our expectations, as passive components like cameras generally consume less electrical power than active components (e.g., RADAR, LiDAR), which need to emit signals to probe the environment.

As shown in Figure 6.28, the latency of the streams transmitted by the cameras is higher than those transmitted by the RADARs due to the larger amounts of data being sent (higher transmission rate). However, the latency of the streams transmitted from the central ECU to the SBW modules remains unchanged.

All considerations regarding the jitter for TTEthernet data remain valid in this case as well.

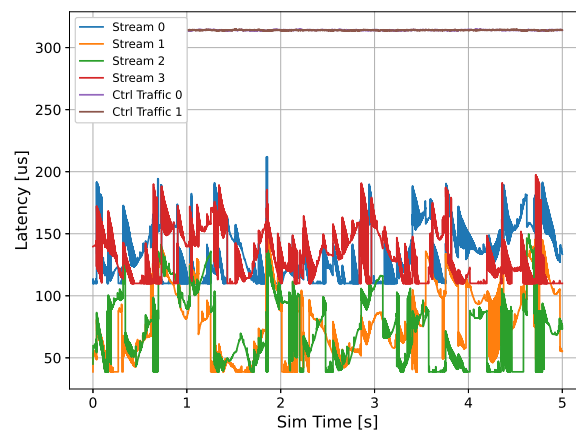


Figure 6.28: Stream Latencies of the Optimized E/E Architecture for Minimized Power Consumption

The load on the CAN bus remains unchanged because the same architecture and

message transmission matrix has been maintained. However, the bandwidth occupation on the backbone (the connection between the switch and the central ECU, where all data is conveyed) has slightly decreased to 62.28% due to the selection of different sensors by the optimizer. This indicates that the communication channel is efficiently utilized, approaching saturation without wasting bandwidth.

Table 6.11: Extended Simulation Results of the Optimized E/E Architecture for Minimized Energy Consumption

Traffic	Src	Dst	BW [Mbps]	Protocol	Max Latency [μ s]	Mean latency [μ s]	Max Jitter [μ s]	Mean Jitter [μ s]
Stream 0	CF3	ECU1	25	AVB Class-A	212.08	130.45	101.97	1.25
Stream 1	ML3	ECU1	6	AVB Class-A	148.55	79.97	98.13	0.89
Stream 2	MR3	ECU1	6	AVB Class-A	149.85	74.43	98.08	1.00
Stream 3	CR0	ECU1	25	AVB Class-A	197.48	128.70	87.49	1.26
Ctrl Traffic 0	ECU1	SBWR	0.05	TTEthernet	315.06	314.07	1.05	0.25
Ctrl Traffic 1	ECU1	SBWL	0.05	TTEthernet	314.91	314.03	0.91	0.25

To summarize, as in the previous case, the network load (both in the CAN and Ethernet domains), latency and jitter requirements, as well as other design criteria such as weight, power consumption, and cost, are all satisfied. This second experiment demonstrates that the entire design exploration framework functions effectively even when the design objective changes.

Further experiments could be conducted to observe how the architecture evolves with different objectives and requirements. However, we aim to set up additional experimental trials to analyze other design parameters. In this direction, the next section provides an analysis performed on the first scenario (maximization of bandwidth) to determine the optimal Hold-Up time in the CAN-to-Ethernet gateway.

6.3.6 Effects of AES Encryption

We further utilized our framework to analyze how the architectural solution changes when the Advanced Encryption Standard (AES) is introduced into the data streams. Cyber-security requirements are becoming increasingly important within in-vehicle networks. In brief, automotive systems use algorithms like RSA (Rivest-Shamir-Adleman), which is asymmetric, for secure key exchange and digital signatures. However, RSA is not suitable for encrypting/decrypting payload data due to its overhead and latency. Therefore, hybrid approaches are often employed: AES handles the encryption/decryption of exchanged data, while RSA is used for generating the keys involved in the process [65].

AES is a symmetric algorithm used for encryption, ensuring the integrity and confidentiality of data transmitted between ECUs within the vehicle. Since the AES we

are considering operates on fixed-size data blocks of 128 bits (16 bytes), an overhead is introduced because the transmitted payload must be a multiple of 16 bytes. Consequently, padding must be applied to fit this size.

We considered this padding in the AVB (Audio Video Bridging) and TTEthernet streams during the design optimization. We conducted two experiments: one focused on maximizing bandwidth at the backbone link and the other on minimizing power consumption.

For the first experiment, which examines the effects of AES encryption during bandwidth maximization at the backbone link, the resulting architecture, sensor mapping, and latencies are illustrated in Figure 6.29, Figure 6.30, and Figure 6.31, respectively.

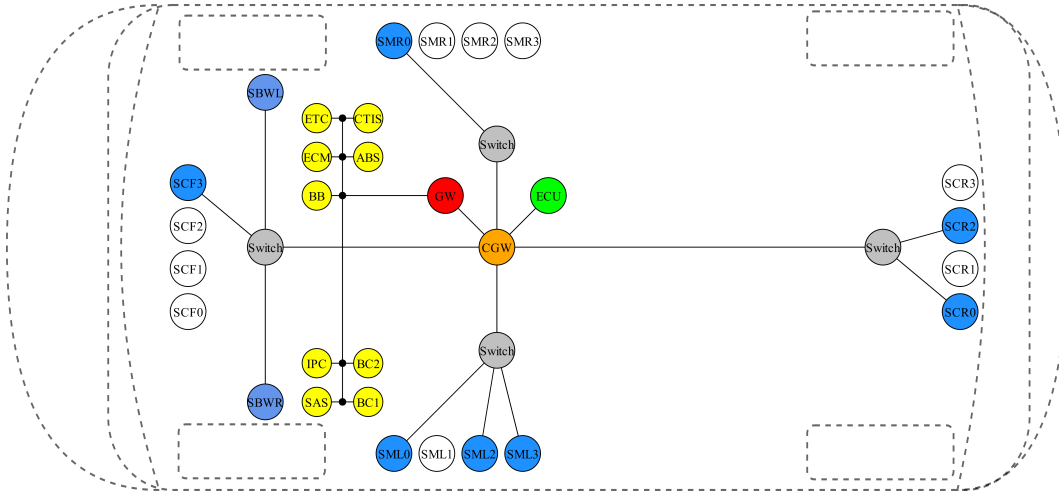


Figure 6.29: Optimized E/E Architecture for Maximized BW Constraint (Colored Components are Those Selected by the Optimizer) when Applying AES Encryption

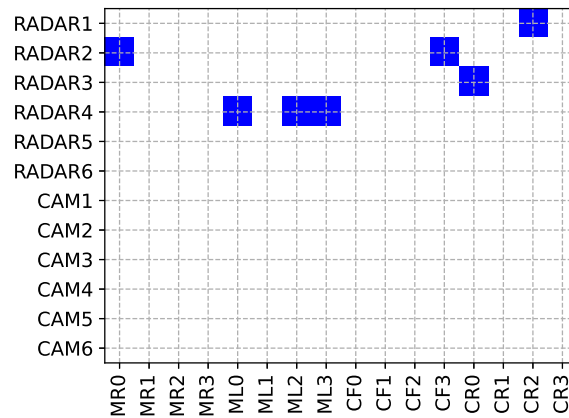


Figure 6.30: Optimized Mapping for Maximized BW Constraint (Only Sensor Library Shown) when Applying AES Encryption

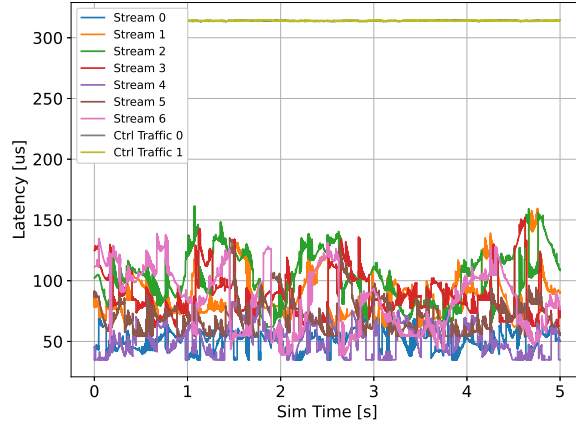


Figure 6.31: Stream Latencies of the Optimized E/E Architecture for Maximized BW Constraint when Applying AES Encryption

Our design solution was found in 9 iterations and 41636 s (approximately 11 hours and 30 minutes), with most of the time attributed to the simulation.

We can see that both the topology and the mapping results differ from those achieved during the same experiment without introducing any encryption, as shown in Figure 6.23 and Figure 6.24. In this case, 7 components instead of 6 are used, and a different combination of library components is selected.

Additionally, the measured bandwidth occupation in the Ethernet backbone is slightly different, at 63.11%.

Regarding CAN bus load and considerations about the latency of TTEthernet streams and AVB streams, the observations are the same as in previous experiments.

In general, the overhead created by the application of AES encryption led the optimizer to select a different combination of sensors to obtain a feasible system solution. Using the same combination of components found in the first experiment (Figure 6.23 and Figure 6.24) with the application of AES encryption results in an unfeasible solution.

Similarly, when applying AES encryption during the minimization of power consumption experiment, the resulting architecture, sensor mapping, and latencies are illustrated in Figure 6.32, Figure 6.33, and Figure 6.34, respectively.

The design solution in this case is found in just 1 iteration (recall that the first experiment is done to find the maximum bandwidth constraint to apply) and 1124 s (approximately 19 minutes). This indicates that solving the initial problem allows us to conduct further experiments with much less design time.

We can see that, in this case as well, both the topology and the mapping results differ

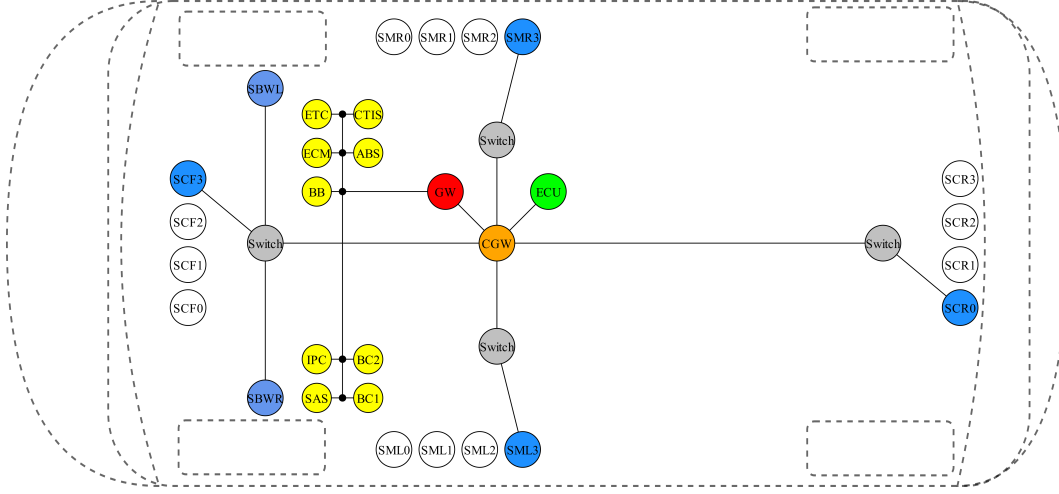


Figure 6.32: Optimized E/E Architecture for Minimized Power Consumption (Colored Components are Those Selected by the Optimizer) when Applying AES Encryption

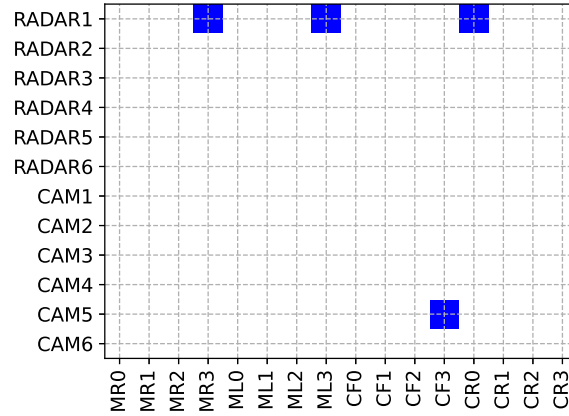


Figure 6.33: Optimized Mapping for Minimized Power Consumption (Only Sensor Library Shown) when Applying AES Encryption

from those achieved during the same experiment without introducing any encryption, as shown in Figure 6.26 and Figure 6.27.

Additionally, the measured bandwidth occupation in the Ethernet backbone is 43.97%. The optimizer selects to use only one camera placed in front of the vehicle, substituting the rear one with a RADAR. Using the previous configuration with the application of AES encryption results in an unfeasible solution.

Similarly, regarding CAN bus load and considerations about the latency of TTEthernet streams and AVB streams, the observations are the same as in previous experiments.

To summarize the outcomes of this experiment, we can say that applying encryption adds an overhead to the data streams, which affects the final architectural results.

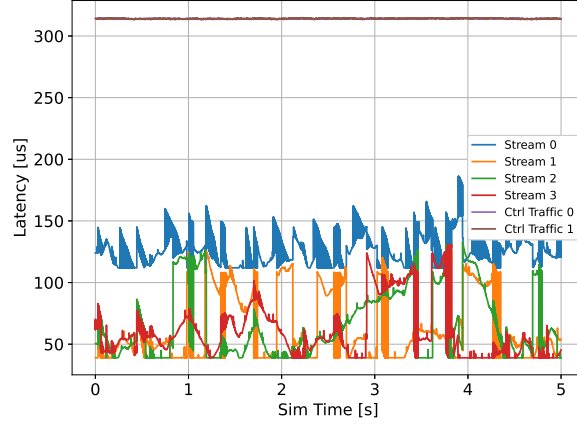


Figure 6.34: Stream Latencies of the Optimized E/E Architecture for Minimized Power Consumption when Applying AES Encryption

Some feasible solutions could become unfeasible when encryption is applied, leading the optimizer to select other solutions. The tool allows us to explore the effects of cybersecurity requirements with only a few lines of code changed instead of hours of analysis. Additionally, it allows us to explore the application of different encryption strategies and to show various architectural possibilities, providing more flexibility during the design phase.

6.3.7 Effects of CAN Aggregation Strategy

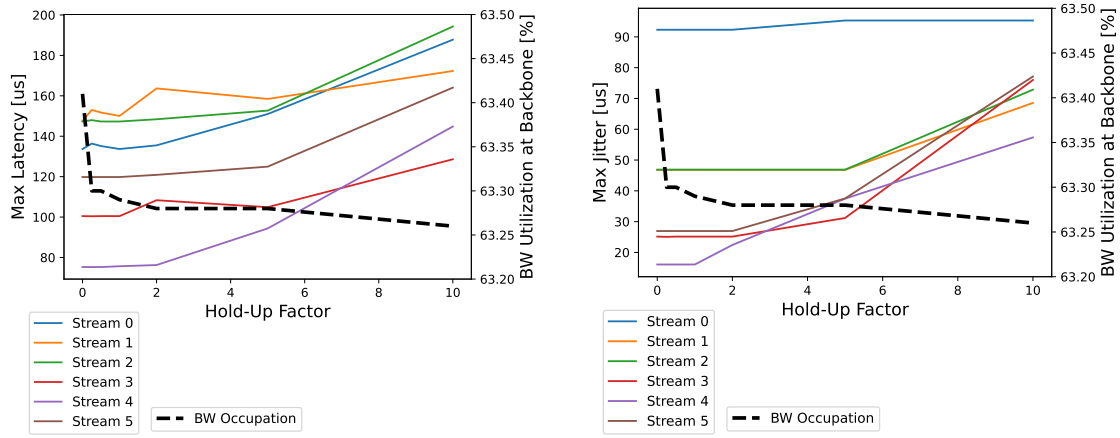
As a further assessment, we use the tool to study the effects of CAN message aggregation (pooling strategy with Hold-Up time). Since the minimum Ethernet payload is 46 bytes and the standard CAN message payload is 8 bytes, direct forwarding of CAN data results in Ethernet payload padding that wastes bandwidth [58]. Given a Hold-Up time, i.e., the maximum acceptable delay for a CAN message, the gateway performs CAN aggregation, releasing frames in groups to the Ethernet domain. More messages are thus assigned to a shared pool and transmitted through Ethernet when the Hold-Up time expires.

The drawback of using an aggregation strategy is an increase in latency and jitter. The experimental effects of CAN aggregation are analyzed for the first scenario, where more bandwidth is used, resulting in a more saturated communication channel. Hold-Up time is set from 0 (CAN message immediately forwarded) to 10 times its transmission period.

In the scenario considered, the reduction in channel utilization on the backbone when increasing the Hold-Up factor is negligible (the channel utilization varies from

63.43% to 63.26%) because the amount of CAN data to be forwarded is small enough that it does not adversely affect the load on the Ethernet backbone. In addition to the delay in CAN messages, the increase in the Hold-Up factor also increases the latency and jitter of the AVB SR-A classes carrying RADAR and camera data, as shown in Fig. 6.35.

For this reason, the best design choice for the gateway is to immediately send CAN messages over the Ethernet domain without using aggregation.



(a) Comparison Between Maximum Latency and Different Hold-Up Factors (b) Comparison Between Maximum Jitter and Different Hold-Up Factors

Figure 6.35: Latency and Jitter Analysis for Different Hold-Up Factor Values

In fact, although TTEthernet frames are unaffected by this behavior (as they transmit in reserved time slots), some AVB frames experience a significant increase in both latency and jitter, as clearly visible from the plots. This results in a deterioration of network performance and behavior.

However, this design choice regarding the Hold-Up factor may not hold true for other architectures [58]. In our case, i.e. the off-road vehicles domain, which typically have distributed architectures, contain fewer ECUs than traditional cars, resulting in less traffic. Converting this data into the Ethernet domain does not impact the total system bandwidth, which is much higher than that used in the CAN domain. Nevertheless, the ability to tune the Hold-Up factor means that our framework can also be applied to other domains (e.g., traditional cars) to perform this type of analysis.

In the next section, we further evaluate our framework by deriving Pareto fronts, which are invaluable for identifying trade-offs and making informed design choices.

6.3.8 Pareto Fronts

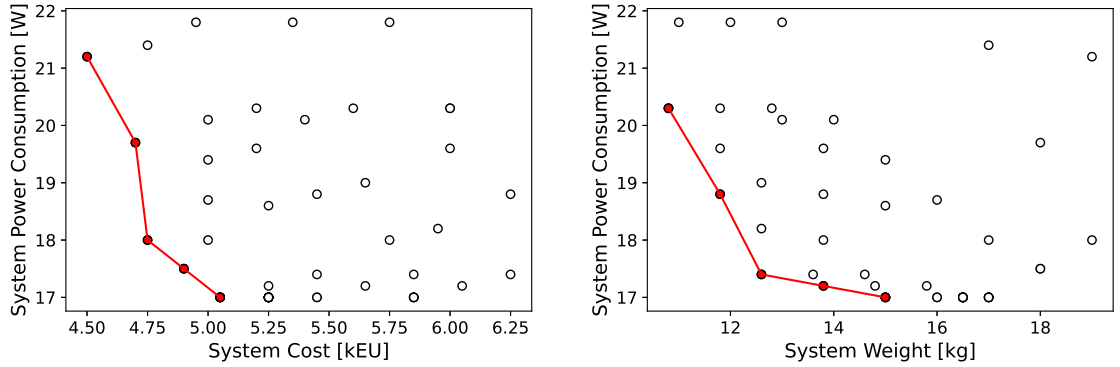
As anticipated, we used our design space exploration framework to derive Pareto fronts, identifying trade-offs between system power consumption and total cost, as well as between system power consumption and total weight.

We conducted this analysis by applying different constraints on weight and cost, and minimizing the system total power consumption. From these results, we derived Pareto fronts, which represent sets of solutions that are non-dominated relative to each other but superior to the rest of the solutions in the search space.

Once again, we emphasize that solving the initial problem related to bandwidth constraints enabled us to perform this Pareto analysis, which would have been very challenging otherwise.

This analysis is crucial from an industrial perspective, as it allows engineers to guide the design not only from a technical standpoint but also by considering other commercial and marketing business processes or requirements that depend on the supply chain.

Initially, we performed a bi-objective analysis to evaluate the trade-offs between power consumption and cost, as well as between power consumption and weight. The outcomes are illustrated in Figure 6.36.



(a) Pareto Analysis: System Power Consumption vs. System Cost. The Red Line is the Pareto Front.

(b) Pareto Analysis: System Power Consumption vs. System Weight. The Red Line is the Pareto Front.

Figure 6.36: Pareto Fronts in the Case Study for Optimizing System Power Consumption

Clearly, reducing the system power consumption results in higher costs for the final solution. Similarly, decreasing the total system weight necessitates accepting higher power consumption. The graphs in Figure 6.36 are instrumental in evaluating which compromises are acceptable in the design process.

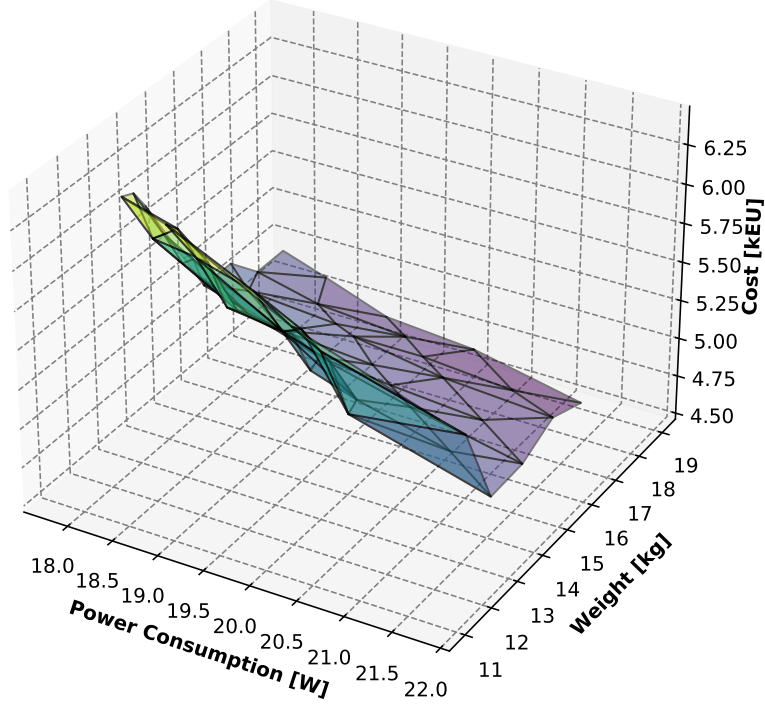


Figure 6.37: Pareto Analysis: 3D Plot of Power Consumption, Weight and Cost

Next, we conducted a multi-objective optimization to derive the Pareto surface, which allowed us to analyze the interactions among system cost, power consumption, and weight. Again, this comprehensive approach provides a holistic view of the trade-offs between these design metrics, facilitating more informed decision-making.

By examining the Pareto surface, we can identify optimal solutions that balance these competing objectives. For instance, reducing power consumption might increase costs, while minimizing weight could lead to higher power usage. Understanding these trade-offs is crucial for making design choices that align with specific project goals and constraints. The visual representation of these trade-offs helps in evaluating which compromises are most acceptable for the overall system performance.

We presented two 3D views in Figure 6.37 and Figure 6.38, where cost is depicted on the z-axis, and power consumption and weight are plotted on the x and y axes. To enhance data visualization, we provided two different projections with the x and y axes flipped.

This dual-projection approach aids in better understanding the relationships among

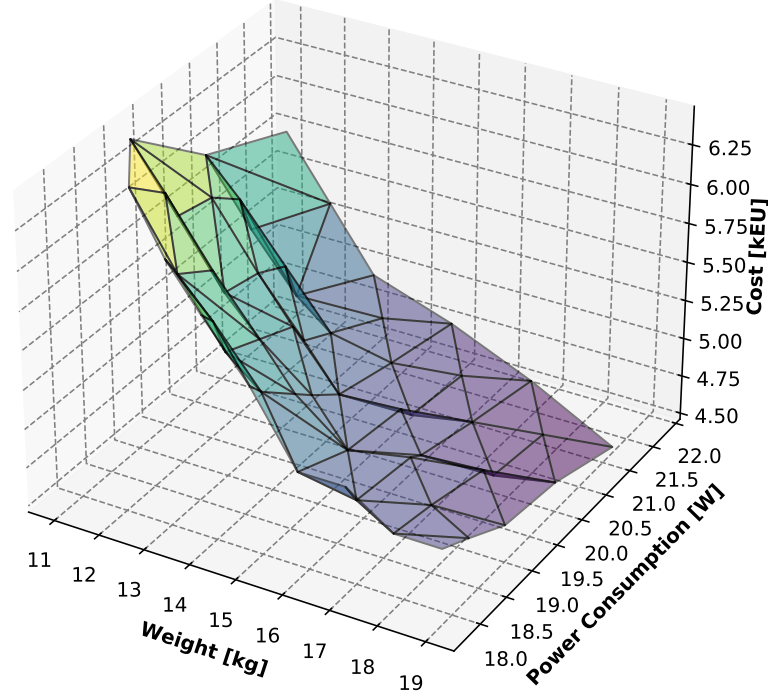


Figure 6.38: Pareto Analysis: 3D Plot of Weight, Power Consumption and Cost

cost, power consumption, and weight, making it easier to interpret the trade-offs between these variables.

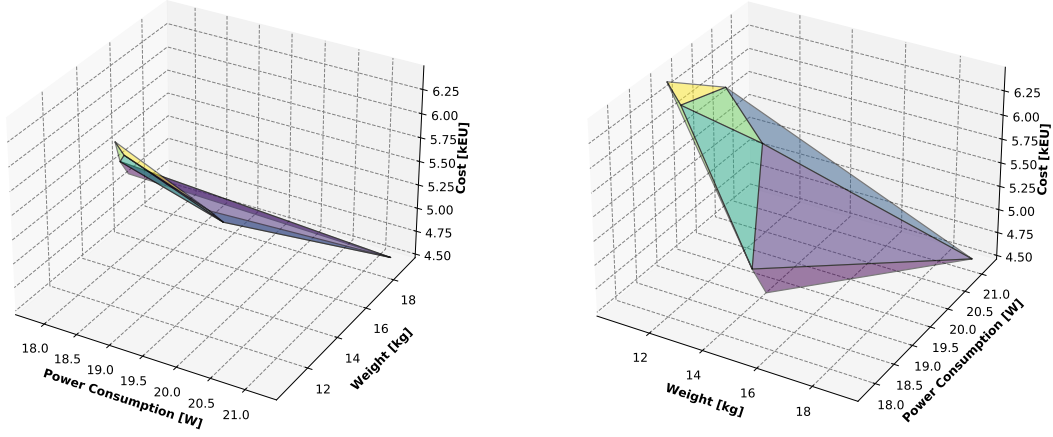
For instance, Figures 6.37 and 6.38 illustrate that achieving lower power consumption and weight comes at the expense of higher costs. Conversely, these figures also demonstrate that reducing costs may result in increased weight and power consumption.

We would like to emphasize that all the solutions presented in this Pareto front are feasible. This is because we established the design constraints and validated the solutions through our framework.

The challenge of this method, which involves varying the constraints within their range of existence using a defined step size, is the potential to overlook solutions that could dominate others due to incomplete simulation of all points. This issue can be mitigated by fine-tuning the step size to a level where the computational cost remains acceptable for the design process.

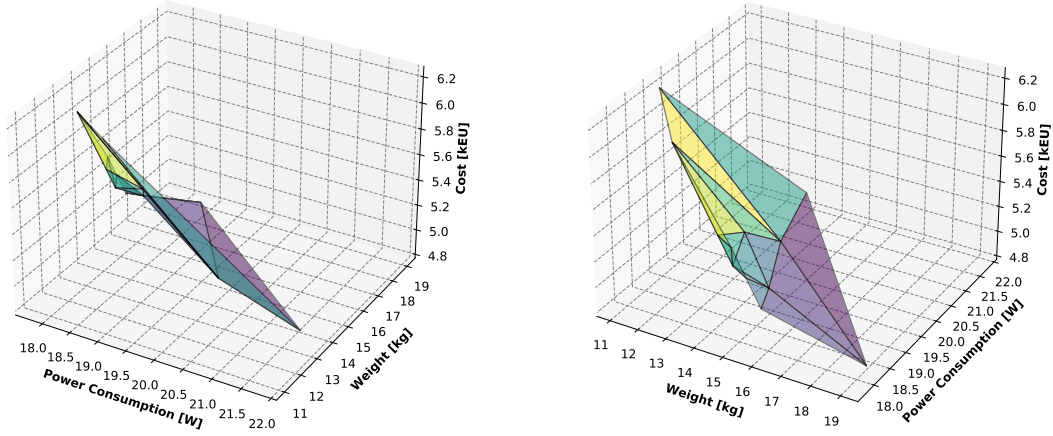
To explore the design space more efficiently, alternative multi-objective techniques can be employed. Figure 6.39 illustrate the Pareto surfaces generated using the NSGA-

II algorithm. In contrast, Figure 6.40 present the Pareto surfaces obtained via Multi-Objective Bayesian Optimization (MOBO).



(a) 3D Pareto Surface: Power Consumption, Weight, and Cost derived using the NSGA-II Algorithm. (b) 3D Pareto Surface: Weight, Power Consumption, and Cost derived using the NSGA-II Algorithm.

Figure 6.39: Comparison of 3D Pareto Surfaces obtained using the NSGA-II Algorithm, highlighting trade-offs between Power Consumption, Weight, and Cost.



(a) 3D Pareto Surface: Power Consumption, Weight, and Cost derived using the MOBO Algorithm. (b) 3D Pareto Surface: Weight, Power Consumption, and Cost derived using the MOBO Algorithm.

Figure 6.40: Comparison of 3D Pareto Surfaces obtained using the MOBO Algorithm, highlighting trade-offs between Power Consumption, Weight, and Cost.

In general, when the problem size is small, deriving Pareto surfaces using a small step size can yield precise results. However, it is essential to carefully tune the step size to balance computational time with maintaining sufficient granularity to control the constraints effectively.

As the problem size increases, for instance, by incorporating more library components or additional constraints, the classical method becomes impractical due to excessive design time.

Building on the initial two experimental evaluations, different objectives can be explored using Pareto fronts to identify further trade-offs. For example, new design constraints could include component size—critical for mechanical installation requirements—or IP protection levels, to examine how system costs vary with increased robustness in harsh environments such as those with dust or sand.

Adopting alternative methods, such as the Multi-Objective Bayesian Optimization (MOBO) or the NSGA-II algorithm, could significantly reduce design time. Although basic implementations of these methods have been provided, exploring improved versions in the future is advisable.

Looking forward, we intend to perform a qualitative scalability analysis to evaluate whether our design methodology can scale to more extensive scenarios. This includes cases where additional sensors are used or the communication link operates with higher bandwidth. Such an analysis is crucial to assessing the extensibility of our design space methodology, not only within the off-road domain but also in broader applications like commercial vehicles.

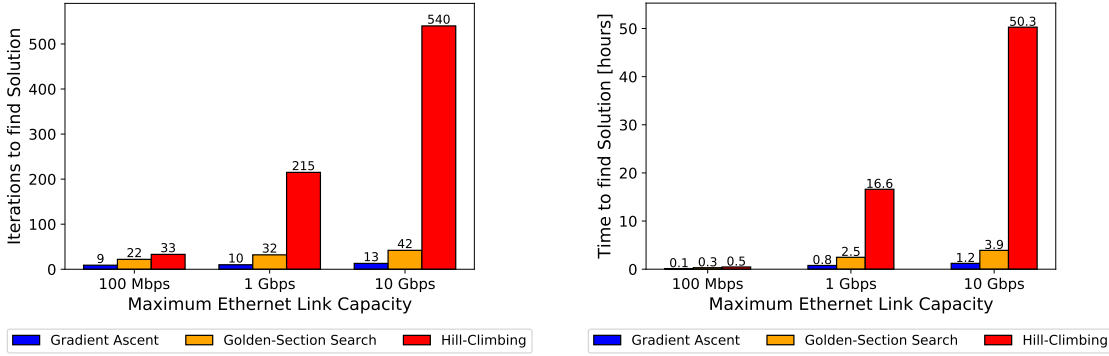
The following section will analyze the total design time, which is influenced by both the number of iterations needed to reach a final solution and the total number of sensors used in the system.

6.3.9 Scalability

As anticipated, the last contribution of this work is a qualitative analysis of the possible scalability of this design methodology in the case of system bandwidth optimization, as there are no characterizations of a similar methodology applied to an off-road scenario in the literature. In particular, we tested the tool for larger scenarios but decreasing the simulation time to 0.5 s (which is the minimum reasonable value to see the feasibility of the solution, otherwise it would be impossible in terms of time to complete the characterization).

In the first analysis, we extended the architectural template to support up to 32

sensors, thus saturating the bandwidth in three different cases by setting a maximum of 100 Mbps, 1 Gbps and 10 Gbps, respectively (the greater the capacity of the link, the greater the amount of data transmitted, and consequently the longer the time required to complete a simulation). We then retrieved the iterations and the total resolution time obtained with our method (gradient ascent with adaptive learning rate) and the golden-section search, and compared them with the hypothetical iterations and the relative time required by a simple linear hill-climbing search (both this approach and a brute force one would be impossible to implement given the large time required for simulations). The iterations required by a linear hill-climbing search are calculated by reasonably assuming that the initial bandwidth constraint is set to a fraction of the maximum link capacity and then incremented to the value found with our tool. The total time for the linear hill-climbing search was calculated by taking the average simulation time obtained with our tool multiplied by the number of iterations. Comparative results for iterations and total resolution time are illustrated in Figure 6.41. Our methodology achieves shorter resolution times and requires fewer iterations compared to the golden-section search, with a substantial advantage over the hill-climbing linear search method. This performance difference becomes increasingly significant as the maximum bandwidth capacity of the system grows.



(a) Maximum Link Capacity vs Total Iterations of Gradient Ascent and Linear Search Methods (b) Maximum Link Capacity vs Resolution Time of Gradient Ascent and Linear Search Methods

Figure 6.41: Scalability Analysis for Different Maximum Link Capacity Scenarios

It is evident that, although the golden-section search remains applicable within acceptable design time limits, the number of required iterations (and consequently, the number of simulations), grows more rapidly compared to the gradient ascent method with an adaptive learning rate. This result underscores the validity of our initial choice of optimization algorithm for determining the maximum bandwidth constraint, affirm-

ing its effectiveness and efficiency.

In the second analysis, we evaluated the maximum resolution time by increasing the number of sensors within a larger scenario and testing two different link capacities.

This approach is advantageous for meeting both immediate requirements, where additional General Safety Regulation (GSR) mandates must be incorporated into the vehicle, and for future projects. As depicted in Figure 6.42, the number of sensors required on vehicles is expected to rise significantly in the next years. This trend underscores the increasing complexity and sophistication of vehicle systems, needing advanced sensor integration to enhance safety, performance, and ensure compliance with evolving regulatory standards.

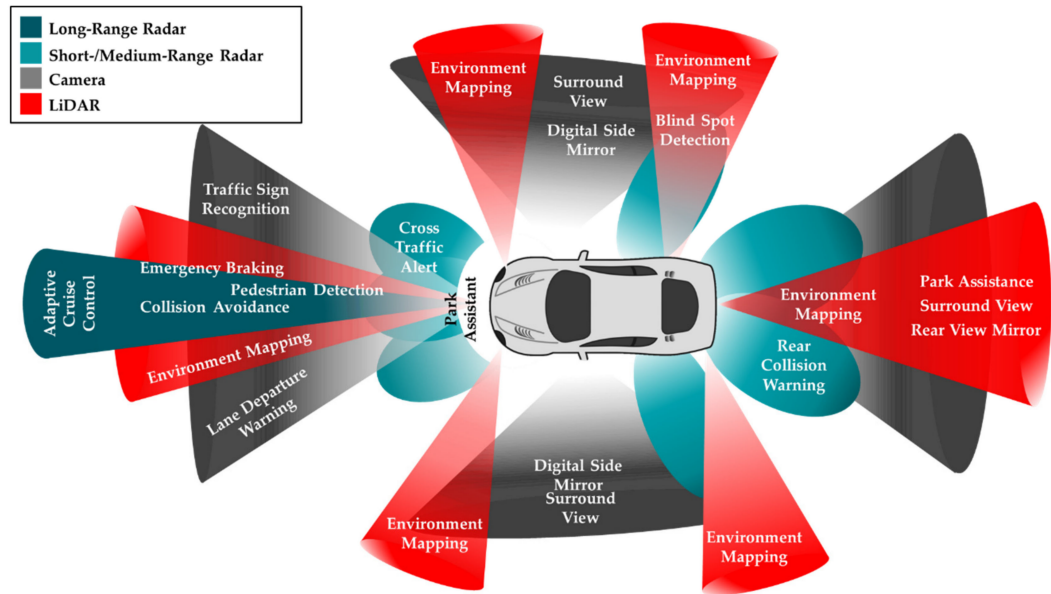


Figure 6.42: Figure from [111]. Sensor Type, Positioning and Relation With Autonomous Driving Functions

In particular, we set the maximum number of sensors in the template from a minimum of 4 to a maximum of 32, and we set the link capacity to 100 Mbps and to 1 Gbps (Fig. 6.43). The total resolution time grows as the number of sensors increases and when the maximum link capacity is higher. However, it is still limited and acceptable from a computational resources perspective, considering that a number of 30 (or more) sensors are required for projects which must satisfy SAE Level 3 for autonomous driving ¹.

¹<https://semiengineering.com/how-many-sensors-for-autonomous-driving/>

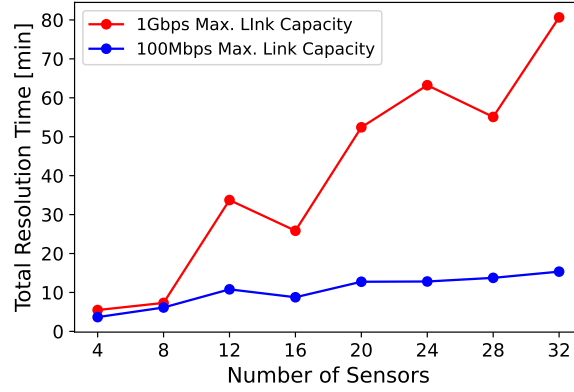


Figure 6.43: Number of Sensors and Connection Capabilities vs. Total Resolution Time

6.4 Discussion

In this third evaluation, we successfully finalized the framework development by implementing the automated constraint reformulation strategy and validating it through various case studies and scenarios. Specifically, we conducted experiments to identify the most effective optimization methodology for reformulating constraints, ultimately selecting the gradient ascent method with an adaptive learning rate. Following this, we examined two case studies: the first focused on optimizing system architecture and implementation by maximizing bandwidth constraints, while the second aimed at minimizing power consumption. These experiments were repeated to analyze the impact of AES encryption, showcasing how it influenced the resulting system architectures.

Additionally, we explored the effects of CAN aggregation to determine the optimal value for the gateway design. We then performed multi-objective optimization using various techniques to assess trade-offs among cost, weight, and power consumption. Finally, we conducted a scalability analysis to evaluate the framework applicability to larger and more complex scenarios.

The next chapter will provide a comprehensive conclusion and an in-depth discussion of the dissertation findings and contributions.

Chapter 7

Conclusions

In this chapter, we start with a summary of the work completed. Next, we present our main results and discuss their impacts. Finally, we provide a discussion on how the framework can be expanded in the future.

7.1 Summary of the Work

At the beginning of this dissertation, we contextualized our work within the problem of architectural design for the Electrical and Electronic (E/E) architectures of off-road vehicles. We highlighted the challenges and considered industry-driven project constraints. These constraints depend on both regulatory requirements and internal project design constraints.

Regarding regulatory requirements, they are defined by the European General Safety Regulation (GSR), which comprises a set of specific requirements stating that a vehicle must be able to perform particular functions to enhance driving safety and comfort to be sold on the market. Our analysis and development focus specifically on the requirements related to the off-road sector.

About system design constraints, we structured our framework to address all project constraints related to industry-driven factors. Specifically, these constraints include time to market (design time), risk (an unsuccessful project could result in significant financial loss), and project feasibility. Considering that a vehicle is not only composed of electrical/electronic systems (even though their role is becoming increasingly important in current and future vehicle deployment), we must also ensure integration with other systems such as mechanical and hydraulic. Consequently, we need to consider not only the cost but also other parameters like total weight and power consumption.

After the introduction, we reviewed related works that address design space exploration not only in the automotive domain but also in other cyber-physical systems. Given that the off-road domain is a niche area, to the best of our knowledge, there are no design-space exploration frameworks specifically tailored to it. Although there are works focused on commercial vehicles that could potentially help us achieve our design goals, they primarily concentrate on the optimal assignment of tasks to hardware components. This approach often overlooks other design goals such as system implementation (component selection) and the optimization of parameters like bandwidth, power consumption, weight, and cost.

Another gap is the lack of comprehensive testing and validation of an optimal solution within the entire architecture. Most techniques focus solely on system design optimization, which might yield feasible solutions analytically but could result in impractical implementations in reality. To address this, some works have introduced simulation to refine design results. However, they often do not discuss the validation and testing of the optimized architecture when integrated into the overall system, which is crucial in our case.

Therefore, we have chosen a design methodology that combines both optimization (based on ArchEx) and simulation (based on CoRE-RG framework), iteratively refining design constraints to converge on a final system design solution. This approach allows us to leverage the advantages of both techniques. The innovative aspect lies in the fact that our simulations validate not only the optimized part of the system but the entire system architecture. It is important to note that, as a system design constraint, we must maintain the existing CAN architecture that manages standard vehicle functionalities while designing a new part of the system to meet the new regulatory requirements.

Next, we conducted an in-depth review of communication standards used for in-vehicle networks, drive-by-wire technologies, architectural topologies, and simulation environments. This analysis is crucial for identifying the most suitable cutting-edge technologies to achieve our design goals. It forms the foundation for creating a custom design template for the optimizer and selecting the appropriate simulator for our purposes.

We then formalized our design methodology, detailing the exploration technique and describing the parameters we evaluate through simulation to determine solution feasibility. Additionally, we outlined our constraints reformulation strategy, which relies on a gradient-based algorithm.

Following the description of our methodology, we provided a detailed account of how we designed the design space exploration framework. This includes an in-depth look

at all the software components, the simulation process, and the interface between the optimization and simulation phases. We also explained how this interface is managed iteratively. Finally, we reported the primary evaluations relevant to the dissertation.

We incrementally tested the tool through different evaluations and phases:

- **First Phase:** This phase focused on testing and validating the initial release of the tool within the context of E/E architecture for off-road vehicles. The optimized solution was generated automatically, while its validation was performed manually (simulation files, the execution of the simulations, and the analysis of the results were all handwritten). Some relevant results from this initial evaluation were published in the article "Design Space Exploration and Validation of Electrical and Electronic System Architectures for Off-Road Vehicles and GSR Requirements," which was presented at the annual conference of the Italian Electronics Society (SIE 2023) in Noto (SR, Italy) from September 6th to 8th, 2023 [13].
- **Second Phase:** In this phase, we enhanced the tool by automating the generation of simulation files, the execution of simulations, and the analysis of results. The tool was applied to a new domain—the design of indoor robotics networks. Additionally, we introduced the concept of constraints reformulation, which became a crucial aspect in the subsequent phase. Select results from this evaluation were published and presented at the IEEE 20th International Conference on Factory Communication Systems in the article "Toward Simulation-Assisted Architecture Design Space Exploration of Indoor Robotics Networks" (WFCS - Toulouse, France, April 17-19, 2024) [14].
- **Third Phase:** This final phase marked the completion of the framework, achieving full automation of the design chain through the integration of the constraints reformulation component. Extensive testing and analysis were conducted using industrially significant case studies. The primary outcomes of this evaluation were published in the open-access journal MDPI Sensors, in the special issue on Design, Communication, and Control of Autonomous Vehicle Systems, under the article "Combining Optimization and Simulation for Next-Generation Off-Road Vehicle E/E Architectural Design" [12].

In the next sections, we present and discuss the main results and impacts of the dissertation.

7.2 Impacts and Main Results

Considering our primary goal, i.e. the development of a framework for design space exploration of E/E architectures in off-road vehicles, we provide a comprehensive tool to achieve this purpose.

In brief, the tool allows for the specification of a system topology through an over-sized template consisting of system components available on the market. It also enables the specification of system design constraints such as cost, weight, power consumption, and bandwidth occupation at backbone. The problem definition, which includes both the topology and the constraints, along with the library of components, can be specified by the designer in simple text files.

Given the project constraints, the optimizer generates a candidate architecture by selecting connections among components and determining which real components must be used to realize the physical system. The simulator then checks if the solution is feasible when integrated into the overall system (including the CAN bus network) and meets our design requirements. If the solution does not meet the requirements, the constraints are reformulated, and the process repeats until convergence is achieved.

In particular, we not only developed the tool from an academic perspective but also guided its development through industry-based requirements and tested it on a significant real-world case study.

Regulatory and Commercial Requirements. To meet the requirements introduced by current regulations, the existing CAN and LIN networks are unable to support the data rates required by new sensors such as radar and cameras, which are essential for developing the required safety driving functions. This is even more critical when considering autonomous driving capabilities. Therefore, we studied which cutting-edge technologies and architectural topologies are desirable to achieve this purpose, selecting the most suitable ones for our system design and using them as the foundation for design space exploration through the framework. Additionally, we conducted a thorough analysis of the GSR requirements, identifying which CAN messages must be routed from the current vehicular CAN domain to the new domain tailored for safety and autonomous driving functionalities. In other words, we design the CAN-Ethernet gateway.

When deploying real vehicles, commercial requirements must be considered, as they directly impact technical design choices. Time-to-market and cost are crucial factors in the overall vehicle design process. Although literature suggests that a complete redesign of the vehicle is highly desirable, it is not industrially feasible due to the high costs and risks involved.

Therefore, we have taken this aspect into consideration by maintaining the existing vehicular CAN architecture and focusing on optimizing the design of the new components. We then use simulation to test not only the optimized architecture but also the overall system, ensuring thorough integration testing.

Integration Requirements. During the design process, several additional aspects must be considered. From a mechanical standpoint, the total weight of the vehicle must not exceed a certain limit, which also restricts the weight of the electronic systems. Additionally, the electrical system has a power budget that must accommodate not only the Electrical and Electronic (E/E) components but also the hydraulic systems (e.g., hydraulic pumps) and mechanical systems (e.g., electro-actuators). Therefore, there is a limit related to the overall power budget that must be satisfied. We included all these constraints within our framework.

Technical Requirements. Finally, the technical requirements specific to the system under design were taken into consideration to determine the feasibility of the solution. In particular, we ensured that the CAN bus load did not exceed 50% to guarantee that all messages were transmitted according to their transmission periods without losing data packets. Additionally, we verified that all AVB and TTEthernet streams were transmitted in the network as expected. We checked that the latency of each stream was under 2 ms. Specifically, for the TTEthernet streams, we also ensured that their jitter was as close to zero as possible.

Development Process Impacts. From an industrial perspective, our tool aims to transcend the traditional design process, which relies heavily on past experience, derivation from existing projects, and designer knowledge. While this approach can be effective in the short term, it becomes increasingly unreliable as system complexity grows [102]. Additionally, it does not facilitate the exploration of different system solutions, leading to an inefficient exploration of the design space. This inefficiency means that alternative solutions, which could enhance other design parameters and improve the overall quality of the product, are often overlooked.

Moreover, the design time using such traditional methods is very high, making it difficult to manage changes during the design phase without significantly impacting the time-to-market. In contrast, a design space exploration framework can identify the same solution, or even a less optimal one but still acceptable, with significantly less time and effort. This efficiency allows for more manageable corrections during the design phase, reducing risks and improving overall control.

Finally, the use of simulation significantly reduces the cost of building physical prototypes, especially when multiple iterations are needed to refine the design. Simulations

provide results that closely mirror real-world conditions, allowing for more precise estimations of system behavior compared to analytical methods, leading to a more efficient and cost-effective design process.

Results. We demonstrated through our framework how the combination of optimization and simulation can lead to efficient exploration of the design space, resulting in feasible solutions validated both analytically and through simulation. Optimization prunes away unfeasible solutions from a large design space, while simulation models real behaviors, such as protocol handshakes and re-transmissions, which are difficult for an optimizer to capture. This iterative combination, with a constraint reformulation strategy, allows us to quickly find satisfying design solutions.

We tested the framework through various case studies. In the second study, we applied it to a different domain—the design of indoor robotic networks. This shows that, while tailored for optimizing off-road vehicle E/E architectures, the framework and its concepts can be adapted to other domains with the necessary modifications.

In the first and third studies, we tested the framework on case studies related to the design of off-road vehicle E/E architectures. We incorporated cutting-edge technologies like zonal-based architectures, drive-by-wire systems, and advanced communication standards, creating a comprehensive, industry-oriented architectural template. We used a component library consisting of real, market-available components and included all relevant design requirements and constraints as parameters.

Furthermore, we used the framework to explore additional design possibilities. We tested architectural changes when cryptographic overhead was introduced to network transmission. We also analyzed the effect of CAN aggregation strategies on network latency and jitter, discussing trade-offs and identifying the optimal design solution for the CAN-Ethernet gateway. We derived both Pareto fronts and surfaces to visualize trade-offs among cost, weight, and system power consumption.

Finally, we conducted a qualitative scalability analysis to assess the framework potential for larger scenarios. We found that, especially with higher link capacity and an increasing number of sensors, the problem resolution time and the number of iterations needed to find solutions were significantly lower compared to linear search methods. This indicates that our tool is capable of scaling to larger scenarios and future applications.

7.3 Discussion and Future Deployments

In this concluding section of the dissertation, we will discuss the limitations of the current work and explore potential solutions to address them in future developments. Additionally, we will propose a strategy for leveraging our tool to facilitate the transition toward fully centralized zonal architectures over the next decade. This approach aims to achieve this evolution with minimal impact on the system, specifically avoiding the need for a complete redesign of the existing E/E network.

7.3.1 Reliability and task Mapping

Our tool is mainly tailored for the design optimization of E/E architectures. It addresses architectural topologies, requirements satisfaction (integration, commercial, and regulatory), and component selection. It also analyzes metrics such as latency, jitter, and network load, along with design parameters like cost, weight, and power consumption. However, some aspects, such as reliability requirements and task mapping on CPUs, have been neglected, as they are already widely explored in other works ([82, 4]).

For instance, reliability aspects can be integrated into the tool to accommodate functional safety requirements like those specified in ISO SAE 26262 [79] or IEC 61508 [34]. Safety Integrity Level (SIL) can be introduced as a design objective by incorporating additional linear constraints.

Task mapping on cores and hardware units can also be integrated into the tool. Generally, this is done after the architecture is defined, posing a design constraint before its development. Alternatively, it can be co-designed, allowing maximum flexibility by adding extra constraints to the optimizer.

7.3.2 Computational Complexity

To enhance the framework and address the time-intensive nature of simulations, particularly with larger scenarios, it is crucial to tackle the simulation time issue. Optimizing the framework from a computational perspective will improve scalability for future deployments and design explorations. This not only increases the efficiency of the tool but also ensures its adaptability and effectiveness as scenarios grow in complexity and scale.

Parallel Computing. A straightforward solution is to employ parallel computing for simulations. This approach allows for optimization without requiring additional hardware.

Dedicated Server. Adopting a dedicated server can significantly speed up both optimization and simulations, thus improving design time.

Cloud Computing. Although it may be costly, especially for researchers and small companies, the expense of running the framework on a cloud server with high computational capacity can be justified and afforded by larger companies. The reduced design time is a significant benefit that justifies the investment in cloud computing.

Evaluate AVB Stream Schedulability Before Simulation. A valuable improvement could involve implementing a preliminary verification of the architecture by evaluating whether all network switches can provide the required Quality of Service (QoS), specifically guaranteed maximum latency, for all AVB streams.

When the optimizer identifies a candidate architecture, it is known which sensors are included in the system and their transmission rates. Therefore, it is possible to analytically approximate the Stream Reservation Protocol (SRP) mechanism of the AVB standard and determine if all switches can schedule the streams.

Although this does not entirely guarantee that the final solution is effective (due to additional traffic from CAN to Ethernet messages and other types of traffic), it allows for the pruning of the design space by eliminating definitely unfeasible solutions. Subsequently, simulations can still be used to verify final or intermediate solutions, reducing the number of required simulations.

Approximation of Latency Constraints Through MILP Formulation Another potential solution is to employ a linear model, even if approximated, which can be embedded in the Mixed-Integer Linear Programming (MILP) optimizer to account for latency constraints in the optimization process. Currently, the framework mainly considers power consumption, cost, and weight constraints, while latency and network load constraints are evaluated through simulations. A precise MILP modeling of all streams in the network (AVB, TTEthernet, Cross Traffic) may be impractical or impossible due to the excessive number of parameters. However, if an approximation can be found, even if not perfectly precise, the optimization process can include some latency and network constraints. While the effectiveness of this solution in improving design time is uncertain, it is a technical direction which can be explored.

7.3.3 Cyber Security Requirements

When considering cybersecurity requirements, additional measures can be integrated into the overall design solution. While our initial approach addresses some security impacts, other aspects, such as introducing diverse encryption standards and implementing

automatic cybersecurity analysis strategies, can further optimize the system towards secure solutions. The ISO/SAE 21434 framework [80] provides organizations in the automotive domain with a structured process to develop cybersecurity-compliant products. A key component of this framework is the **Threat Analysis and Risk Assessment (TARA)** methodology, which allow designers to identify vulnerabilities within the Electrical/Electronic (E/E) system and to implement countermeasures against external cyberattacks to safeguard the system. The TARA Process Steps (as defined by ISO/SAE 21434) are the following:

1. **Asset Identification:** Identify the key items in the system (e.g., ECUs, sensors, SW components, etc.).
2. **Definition of Cybersecurity Properties:** Define relevant cybersecurity properties, such as confidentiality, integrity, and availability (CIA), for each identified item.
3. **Damage Scenarios and Impact Rating:** Identify potential damage scenarios that could pose risks, such as safety hazards to people, financial losses, operational disruptions, or privacy violations. Evaluate and rate the impacts based on these factors.
4. **Threat Scenario Identification:** Outline scenarios in which an external attacker could exploit vulnerabilities to access the system.
5. **Attack Path Analysis and Feasibility Rating:** Analyze possible methods attackers might use to execute threats, assessing their feasibility based on criteria such as time, expertise, knowledge, opportunity windows, and required equipment.
6. **Risk Value Determination:** Calculate the risk value using a feasibility and impact matrix to prioritize risks.
7. **Risk Treatment Decision:** For risks exceeding acceptable thresholds, implement countermeasures to mitigate them. Repeat the analysis to ensure risks are reduced to an acceptable level.

To better understand the process, we go on with a simple practical example.

Let us assume the system includes a telematic unit to facilitate Over-The-Air (OTA) updates for vehicle software (Figure 7.1). We analyze a scenario in which an attacker gains access to the telematic unit software and manipulates it to send malicious messages to the Engine Control Module (ECM).

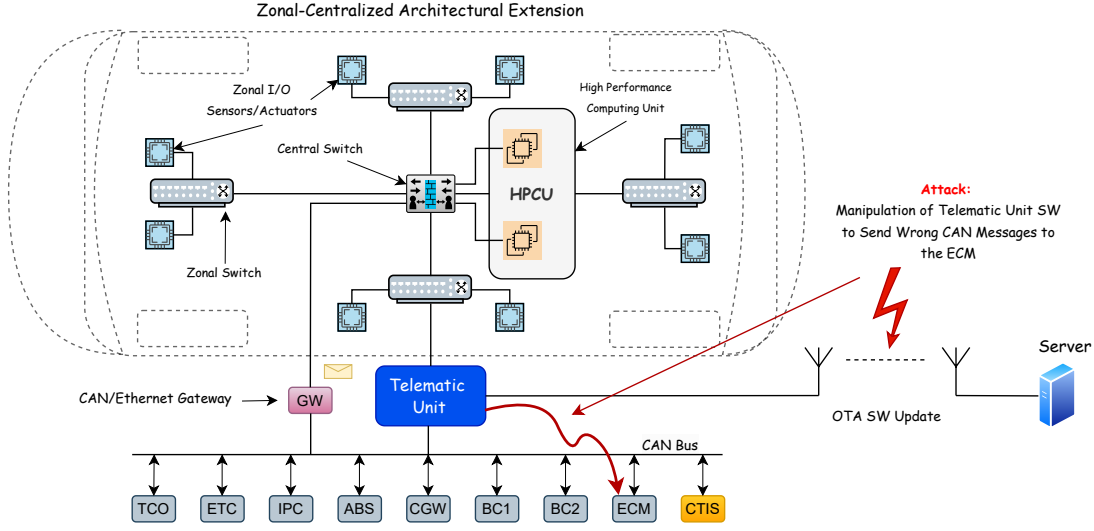


Figure 7.1: TARA: Attack Scenario

Asset Identification: For example, in this scenario, we can simply consider the following assets:

- The radio link serves as the communication channel between the operational server and the vehicular platform.
- The antenna is the physical component used by the telematic unit to transmit and receive data.
- The telematic unit is the hardware responsible for managing diagnostic and telematic services.
- The cryptographic algorithm protects the communication between the server and the telematic unit.

The **Definition of Cybersecurity Properties** for each asset is reported in Figure 7.2.

ID	Asset	Security Properties (CIA)
ID_ASS_001	Telematic Unit SW	I, A
ID_ASS_002	Antenna	I, A
ID_ASS_003	Radio Link	I, A
ID_ASS_004	Cryptographic Algorithm	C, I, A

Figure 7.2: Definition of Cybersecurity Properties

Once damage scenarios associated with specific assets have been identified, they are assessed based on the previously defined properties. For each scenario, the overall impact rating is determined by selecting the highest value among the individual ratings (Figure 7.3).

ID	Damage Scenario	Safety	Financial	Operational	Privacy	Total Max{.}	Asset Linked
ID_DMG_001	Accident due to telematic unit SW manipulation	Severe	Severe	Severe	Minor	Severe	ID_ASS_001
ID_DMG_002	Loss of vehicle functionality due to deletion of telematic unit SW	Minor	Minor	Major	Minor	Major	ID_ASS_001

Figure 7.3: Subset of Damage Scenarios and Impact Rating

The resulting ratings for damage scenarios are utilized in the final stage of the process to determine the overall risk value.

Similarly, the subsequent step involves identifying the threat scenarios, analyzing the attack paths for each scenario, and evaluating the attack feasibility. Continuing with the example, an identification of threat scenarios is illustrated in Figure 7.4.

ID	Threat Scenario	Damage Scenario Linked	Asset Linked	Security Property
ID_THR_001	Attacker manipulate telematic Unit SW to send malicious	ID_DMG_001	ID_ASS_001	I - Integrity
ID_THR_002	Attacker delete telematic Unit SW	ID_DMG_002	ID_ASS_001	A - Availability

Figure 7.4: Threat Scenario Identification (Subset)

Once the threat scenarios are identified, an attack path analysis must be conducted, and a corresponding feasibility rating should be assigned to each step of the attack. As shown in Figure 7.5, the attack feasibility for the threat scenario *ID_THR_001* is presented.

Finally, the **Risk Value Determination** must be completed. The risk value is derived by intersecting the impact rating of the damage scenario with the corresponding feasibility rating of the attack. To ensure the system meets cybersecurity safety requirements, the risk value must be less than or equal to 2, as illustrated in Figure 7.6.

In this scenario, the risk value is 3, necessitating the implementation of countermeasures. For instance, a direct connection between the telematic unit and the Ethernet or CAN domain is not recommended. Instead, routing the communication through a gateway is a more secure approach. The gateway can act as a filter, identifying and rejecting erroneous CAN messages that might otherwise alter the real-time behavior of the Engine Control Unit.

7.3. Discussion and Future Deployments

Attack Step	Time	Expertise	Knowledge	Windows of Opportunity	Equipment	Total Min{.}
Attacker steel cryptographic key	Medium	Low	Low	Low	Low	Low
Attacker read data traffic during an OTA SW Update	High	Medium	Medium	Low	Low	Low
Attacker manipulate telematic unit SW	Medium	Medium	Medium	Low	High	Low
Attacker update SW to send malicious messages to ECM	High	High	Medium	Low	Medium	Low
TOTAL FEASIBILITY	-	-	-	-	-	Low

Figure 7.5: Attack Path Analysis and Feasibility Rating for *ID_THR_001*

Impact	Attack Feasibility	Very Low	Low	Medium	High
		1	3	4	5
Severe		1	3	4	5
Major		1	2	3	4
Minor		1	2	2	3
Negligible		1	1	1	1

Figure 7.6: Risk Value Determination

An implementation of this mitigation is depicted in Figure 7.7

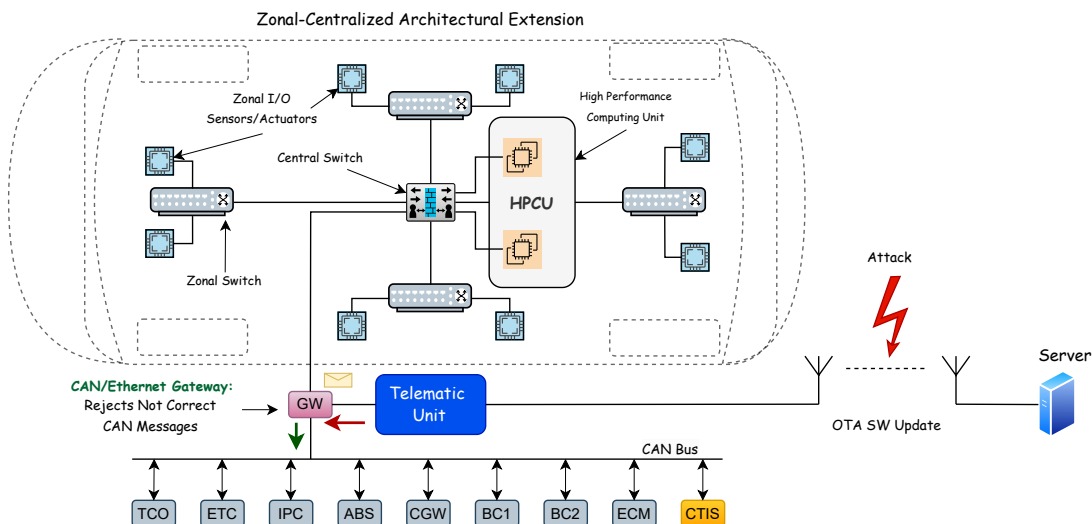


Figure 7.7: TARA: Attack Scenario with Countermeasure

Next, the feasibility rating for the attack should be re-assessed, incorporating the countermeasure of the ability of the gateway to filter unwanted messages. This updated evaluation reflects the increased difficulty of executing the attack due to the gateway

which rejects malicious communications, as shown in Figure 7.8 and Figure 7.9.

Attack Step	Time	Expertise	Knowledge	Windows of Opportunity	Equipment	Total Min{.}
Attacker steel cryptographic key	Medium	Low	Low	Low	Low	Low
Attacker read data traffic during an OTA SW Update	High	Medium	Medium	Low	Low	Low
Attacker manipulate telematic unit SW	Medium	Medium	Medium	Low	High	Low
Attacker update SW to send malicious messages to ECM	High	High	Medium	Very Low	Medium	Very Low
TOTAL FEASIBILITY	-	-	-	-	-	Very Low

Figure 7.8: Attack Path Analysis and Feasibility Rating for *ID_THR_001* with Countermeasure

Attack Feasibility		Very Low	Low	Medium	High
Impact	Severe	1	3	4	5
	Major	1	2	3	4
	Minor	1	2	2	3
	Negligible	1	1	1	1

Figure 7.9: Risk Value Determination with Countermeasure

The inclusion of the gateway reduces the windows of opportunity rating. Even if the telematic unit software is compromised, the gateway intercepts and rejects unwanted messages, ensuring they do not reach the CAN network.

It is important to note that the presented example is simplified and omits many intricate details. In general, cybersecurity analyses consider hundreds of damage scenarios. Moreover, the number of threat scenarios, along with their attack paths and feasibility assessments, grows significantly with the increase in identified assets.

In real-world situations, conducting a comprehensive cybersecurity analysis requires extensive expertise, substantial manual effort, and significant time to ensure all potential vulnerabilities are adequately addressed.

A notable limitation of this work is the absence of these concepts within the design framework. To address this, it is recommended to explore future enhancements that incorporate automatic (or semi-automatic) mechanisms for performing the TARA process. Such advancements would significantly reduce the need for manual effort, saving considerable time and resources while improving overall efficiency.

7.3.4 Transition Towards Fully centralized Architectures

In our dissertation, we have discussed that automotive E/E architectures are transitioning to zonal-centralized configurations. The increasing levels of autonomy and advanced driver assistance features in vehicles necessitate the adoption of new standards to handle higher data volumes from sensors [85]. Additionally, centralizing vehicular functions is essential to enhance scalability, safety, and cybersecurity. Although a complete redesign of the in-vehicle network would be ideal, it is not commercially viable. Therefore, a smooth transition towards fully zonal-centralized architectures is advisable.

Our tool, as depicted in Figure 7.10, can simply accommodate this process.

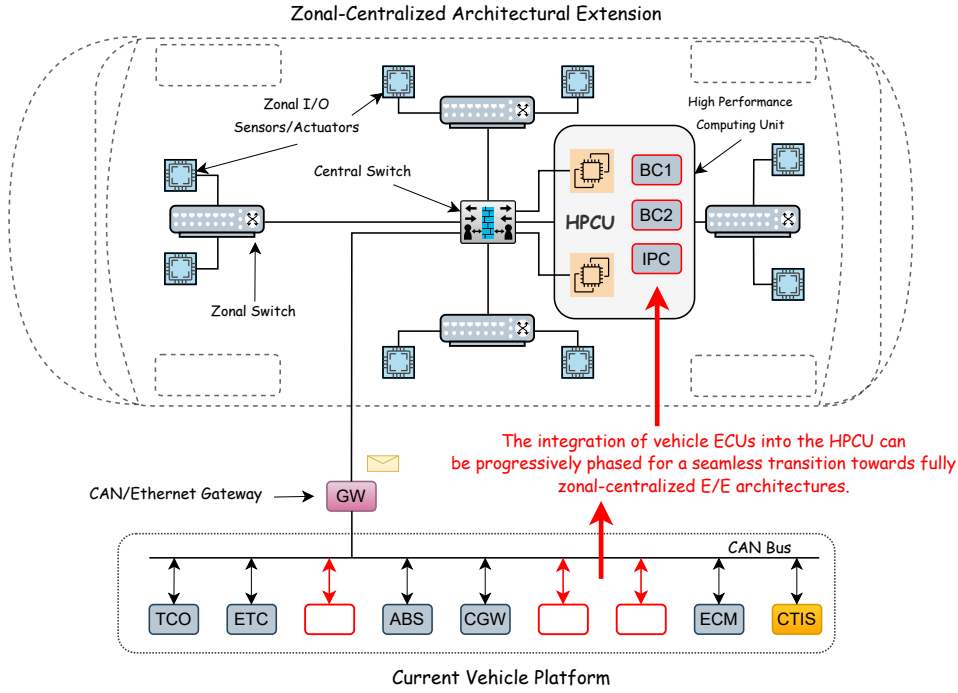


Figure 7.10: Conceptual Smooth Transition Towards Fully Zonal-Centralized Architectures

When ECU functionalities from the current architecture are transferred to the central processing unit (HPCU), the design impacts, such as latencies and network loads, can be easily estimated through our framework. By modeling the new streams required to accommodate these functionalities in the new domain, the tool can find feasible architectural solutions that meet design constraints. This process allows for controlled exploration of design space changes as we perform transition towards fully zonal-centralized

architectures. A final assessment, including a comparative analysis between our solution and other industrial solutions, as well as physical validation, is advisable. Due to the lack of availability of a physical prototype, this type of analysis is postponed to the future. In conclusion, design space exploration offers several benefits for the industrial design of E/E architectures. Our tool models system design requirements through the optimizer and validates architectural solutions through simulation, providing accurate modeling of physical behaviors. The role of simulation is not only to validate the integration of design solutions but also to guide the optimizer and refine constraints to converge on the final design. From an industrial perspective, this methodology goes beyond traditional design processes, which lack flexibility and struggle to manage the design challenges posed by future deployments.

Acronyms

AADL Architecture Modeling and Description Language. 12

ABS Automatic Braking System. 34, 99

ADAS Advanced Driver Assistance Systems. 1, 10, 11, 16, 19, 20, 28, 36, 37, 39, 59, 96, 99, 121, 123

ADCU Advanced Driving Assistance Control Unit. 46, 47, 60, 73, 75, 83, 97, 120

AEES Automotive Electrical and Electronic System. 75

AES Advanced Encryption Standard. 131, 132, 133, 134

AIF Alcohol Interlock Facilitation. 1

API Application Programming Interface. 93

AQOSA Automated Quality-driven Optimization of Software Architecture. 14

ASR Anti-Split Regulation. 34

AVB Audio Video Bridging. 5, 6, 12, 28, 30, 31, 32, 33, 38, 39, 40, 59, 60, 69, 71, 79, 81, 83, 87, 99, 102, 103, 120, 121, 122, 123, 124, 128, 132, 133, 134, 136, 149

BAG Bandwidth Allocation Gap. 29

BBW Brake-by-Wire. 20, 34

BC Body Computer. 2, 20

BSIS Blind Spot Information System. 1

CAN Controller Area Network. vii, 2, 5, 6, 16, 17, 22, 23, 24, 28, 38, 40, 42, 48, 58, 69, 71, 80, 85, 87, 88, 89, 99, 101, 119, 120, 121, 123, 124, 128, 130, 131, 135, 136, 146, 148, 149, 150

- CAPL** Communication Access Programming Language. 38
- CBS** Credit Based Shaper. 32, 39, 40, 121, 122, 128
- CDT** Control Data Traffic. 33
- COTS** Commercial-Off-The-Shelf. 15, 71, 74
- CPS** Cyber-Physical System. 3, 5, 9, 12, 13, 57
- CPU** Central Processing Unit. 59, 151
- CQF** Cyclic Queuing and Forwarding. 39
- CSMA-CA** Carrier Sense Multiple Access - Collision Avoidance. 26
- CSMA-CD** Carrier Sense Multiple Access - Collision Detection. 22, 28
- CSMS** Cyber Security Management System. 1
- CTIS** Central Tyre Inflation System. 2, 101, 123
- DBW** Drive-by-Wire. 2, 19, 20, 34, 41, 42, 69, 120
- DCU** Domain Control Unit. 36
- DDAW** Driver Drowsiness & Attention Warning. 1
- DL** Differential-Lock. 2
- DSE** Design Space Exploration. 3, 4, 9, 10, 11, 13, 41, 71, 119
- E/E** Electrical and Electronic. 1, 2, 3, 4, 6, 7, 9, 11, 15, 16, 19, 34, 36, 38, 39, 40, 41, 45, 46, 50, 51, 55, 57, 69, 71, 73, 77, 95, 96, 102, 117, 119, 145, 147, 149, 150, 151, 158, 159
- E2E** End-to-End. 33, 39, 107, 108, 109, 112, 113, 116
- ECM** Engine Control Module. 2, 20, 34, 99
- ECU** Electronic Control Unit. 10, 21, 25, 28, 35, 36, 41, 46, 48, 51, 80, 83, 85, 97, 99, 100, 101, 120, 121, 123, 128, 130, 131, 158
- ECUs** Electronic Control Units. vii, 2, 10, 11, 12, 20, 21, 22, 28, 35, 36, 46, 121, 131, 136

ETC Electronic Transmission Control. 2, 20

FIFO First In First Out. 81

GSR General Safety Regulation. vii, 1, 5, 96, 101, 121, 123, 128, 143, 145, 148

GUI Graphical User Interface. ix, 80, 86, 88

HPCU High-Performance Computing Unit. 11, 37, 46, 60, 83, 120, 158

HW Hardware. 82, 89

I/O Input/Output. 36, 37

IC Instrument Cluster. 20

ID Identifier. 32, 88

IEEE Institute of Electrical and Electronics Engineers. 30

ILP Integer Linear Programming. 9, 10, 12, 14

IP Internet Protocol. 38

ISA Intelligent Speed Assist. 1

LANs Local Area Networks. 30

LIN Local Interconnect Network. 2, 20, 21, 22, 28, 148

LVDS Low-Voltage Differential Signaling. 27, 41

MAC Medium Access Control. 31, 32, 81, 84, 87

MILP Mixed-Integer Linear Programming. 5, 13, 14, 43, 73, 76, 77, 78, 97, 103, 106, 108, 117

MIP Mixed-Integer Programming. 11

MOIS Moving-Off Information Signal. 1

MOST Media Oriented Systems Transport. 26, 27

OEMs Original Equipment Manufacturers. 1, 35

ORV Off-Road Vehicle. 1, 41, 69

OS Operating System. 12

OSATE Open Source AADL Tool Environment. 12

PCFs Protocol Control Frames. 30

PLC Power-Line Communication. 27, 28

PTO Power-Take-Off. 2, 99

PTP Precision Time Protocol. 30

QoE Quality of Experience. 105

QoS Quality of Service. 28, 32, 105, 107, 117

REV Reverse Detection. 1

RQMTS Requirements. 2

RSA Rivest-Shamir-Adleman. 131

SAE Society of Automotive Engineers. 19, 22, 26, 102, 123, 143

SAOL System Architecture Optimization Language. 14

SBW Steer-by-Wire. 20, 34, 121, 123, 128, 130

SMT Satisfiability Modulo Theory. 11, 12, 14

SRP Stream Reservation Protocol. 30, 32, 38, 81, 87

SW Software. 73, 89

TAS Time-Aware Shaper. 39

TBW Throttle-by-Wire. 34

TC Transfer Case. 2

TCP Transmission Control Protocol. 38

TDM Time Division Multiplexing. 26

TDMA Time Division Multiple Access. 25, 29

TPMS Tyre pressure monitoring Heavy duty. 1

TSN Time-Sensitive Networking. 10, 12, 30, 33, 39, 40, 79

TT Time-Triggered. 5, 6, 9, 12, 23, 28, 29, 38, 39, 40, 69, 71, 79, 100, 120, 121, 122, 128, 130, 132, 133, 134, 136, 149

UDP User Datagram Protocol. 38

URLLC Ultra-Reliable and Low-Latency Communication. 105

VSC Vehicle Stability Control. 34

ZCU Zone Control Unit. 37

Bibliography

- [1] Listserv 16.5 - listserv archives at list.etsi.org. <https://list.etsi.org/scripts/wa.exe?HOME>. (Accessed on 06/30/2023).
- [2] Giuliana Alderisi, Alfio Caltabiano, Giancarlo Vasta, Giancarlo Iannizzotto, Till Steinbach, and Lucia Lo Bello. Simulative assessments of ieee 802.1 ethernet avb and time-triggered ethernet for advanced driver assistance systems and in-car infotainment. In *2012 IEEE Vehicular Networking Conference (VNC)*, pages 187–194, 2012.
- [3] Aldeida Aleti, Stefan Bjornander, Lars Grunske, and Indika Meedeniya. Archeopterix: An extendable tool for architecture optimization of aadl models. In *2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, pages 61–71, 2009.
- [4] Hadi Askaripoor, Morteza Farzaneh, and Alois Knoll. E/e architecture synthesis: Challenges and technologies. *Electronics*, 11:518, 02 2022.
- [5] Hadi Askaripoor, Thilo Mueller, and Alois Knoll. E/e designer: a framework to design and synthesize vehicle e/e architecture. *IEEE Transactions on Intelligent Vehicles*, pages 1–18, 2023.
- [6] Audi. Audi technology portal. Accessed: 2022-01-10.
- [7] Victor Bandur, Gehan Selim, Vera Pantelic, and Mark Lawford. Making the case for centralized automotive E/E architectures. *IEEE Transactions on Vehicular Technology*, 70(2):1230–1245, 2021.
- [8] Haytham Baniabdelghany, Roman Obermaisser, and Ala Khalifeh. Extended synchronization protocol based on ieee802.1as for improved precision in dynamic and asymmetric tsn hybrid networks. pages 1–8, 06 2020.

- [9] Sagar Behere and Martin Torngren. A functional architecture for autonomous driving. In *First International Workshop on Automotive Software Architecture*, pages 3–10, 2015.
- [10] Lucia Lo Bello. Novel trends in automotive networks: A perspective on Ethernet and the IEEE Audio Video Bridging. In *Emerging Technology and Factory Automation*, 2014.
- [11] Karsten Berns. Off-road robotics - perception and navigation. In *11th International Conference on Informatics in Control, Automation and Robotics*, volume 02, pages IS–9–IS–11, 2014.
- [12] Cristian Bianchi, Rosario Merlino, and Roberto Passerone. Combining optimization and simulation for next-generation off-road vehicle e/e architectural design. *Sensors*, 24(15), 2024.
- [13] Cristian Bianchi, Rosario Merlino, and Roberto Passerone. Design space exploration and validation of electrical and electronic system architectures for off-road vehicles and gsr requirements. In Carmine Ciofi and Ernesto Limiti, editors, *Proceedings of SIE 2023*, pages 369–378, Cham, 2024. Springer Nature Switzerland.
- [14] Cristian Bianchi, Ayub Shah, Chiara Marangoni, and Roberto Passerone. Toward simulation-assisted architecture design space exploration of indoor robotics networks. In *Proceedings of the 20th IEEE International Conference on Factory Communication Systems*, WFCS 2024, Toulouse, France, April 17–19, 2024.
- [15] Arash Bozorgchenani, Farshad Mashhadi, Daniele Tarchi, and Sergio Salinas Monroy. Multi-objective computation sharing in energy and delay constrained mobile edge computing environments. *IEEE Trans. on Mob. Comp.*, 2020.
- [16] Longbiao Chen, Dingqi Yang, Michele Nogueira, Cheng Wang, Daqing Zhang, et al. Data-driven C-RAN optimization exploiting traffic and mobility dynamics of mobile users. *IEEE Transactions on Mobile Computing*, 2020.
- [17] Federico D’Aniello, Andreas Ott, and Andrea Baschirotto. Supply line embedded communication in automotive sensor/actuator networks. In *International Symposium on Circuits and Systems*, pages 1–5, 2021.
- [18] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

- [19] Marco Di Natale and Alberto Luigi Sangiovanni-Vincentelli. Moving from federated to integrated architectures in automotive: The role of standards, methods and tools. *Proceedings of the IEEE*, 98(4):603–620, 2010.
- [20] Yifan Duan, Changyang She, Guodong Zhao, and Tony QS Quek. Delay analysis and computing offloading of urllc in mobile edge computing systems. In *10th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6. IEEE, 2018.
- [21] EU. Supplementing Regulation (EU) 2019/2144 of the European Parliament and of the Council by laying down detailed rules concerning the alcohol interlock installation facilitation in motor vehicles and amending Annex II to that Regulation, 2021.
- [22] EU. Supplementing Regulation (EU) 2019/2144 of the European Parliament and of the Council by laying down detailed rules concerning the ISA and amending Annex II to that Regulation, 2021.
- [23] EU. Supplementing Regulation (EU) 2019/2144 of the European Parliament and of the Council by laying down detailed rules concerning the specific test procedures and technical requirements for the type-approval of motor vehicles with regard to their driver drowsiness and attention warning systems and amending Annex II to that Regulation, 2021.
- [24] John Finn, Pierluigi Nuzzo, and Alberto Sangiovanni-Vincentelli. A mixed discrete-continuous optimization scheme for cyber-physical system architecture exploration. In *International Conference on Computer-Aided Design*, pages 216–223, 2015.
- [25] Vector Informatik GmbH. Canoe. Accessed: 2024-01-10.
- [26] A. Goyal and A. Thakur. An overview of drive by wire technology for automobiles. In *International Conference on Automation, Computational and Technology Management*, 2019.
- [27] Sebastian Graf, Michael Glab, Jürgen Teich, and Christoph Lauer. Multi-variant-based design space exploration for automotive embedded systems. In *Design, Automation & Test in Europe*, pages 1–6, 2014.

- [28] Marco Haeberle, Florian Heimgaertner, Hans Loehr, Naresh Nayak, Dennis Grewe, Sebastian Schildt, and Michael Menth. Softwarization of automotive E/E architectures: A software-defined networking approach. In *IEEE Vehicular Networking Conference*, pages 1–8, 2020.
- [29] Jia Hao, Jiechang Wu, and Chaoyou Guo. Modeling and simulation of can network based on opnet. In *2011 IEEE 3rd International Conference on Communication Software and Networks*, pages 577–581, 2011.
- [30] Benjamin C. Heinrich, Thorsten Luettel, and Hans-Joachim Wuensche. A new control architecture for MuCAR. In *IEEE Intelligent Vehicles Symposium*, pages 1861–1866, 2017.
- [31] Robert Höttger, Harald Mackamul, Andreas Sailer, Jan-Philipp Steghöfer, and Joerg Tessmer. App4mc: Application platform project for multi- and many-core systems. *it - Information Technology*, 59, 11 2017.
- [32] Yinjia Huo, Wei Tu, Zhengguo Sheng, and Victor C.M. Leung. A survey of in-vehicle communications: Requirements, solutions and opportunities in IoT. In *2nd World Forum on Internet of Things*, pages 132–137, 2015.
- [33] Ahmed Hussein, Pablo Marín-Plaza, David Martín, Arturo de la Escalera, and José María Armingol. Autonomous off-road navigation using stereo-vision and laser-rangefinder fusion for outdoor obstacles detection. In *Intelligent Vehicles Symposium*, 2016.
- [34] IEC. Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements, 2010.
- [35] ISO/IEC/IEEE. ISO/IEC/IEEE 15288:2023 Systems and software engineering — System life cycle processes, 2023.
- [36] Shriram Jadhav and Deepak Kshirsagar. A survey on security in automotive networks. In *Fourth International Conference on Computing Communication Control and Automation*, 2018.
- [37] Prachi Joshi, G. Vedahari Narasimhan, Haibo Zeng, Sandeep K. Shukla, Chungwei Lin, and Huafeng Yu. Design space exploration for deterministic Ethernet-based architecture of automotive systems. In *International High Level Design Validation and Test Workshop*, pages 53–61, 2016.

- [38] Azyan Yusra Kapi, Mohd Shahrizal Sunar, and Zeyad Abd Algfoor. Summary of pathfinding in off-road environment. In *6th International Conference on Interactive Digital Media*, pages 1–4, 2020.
- [39] Eunchong Kim, Kanghyun Park, Hunmin Yang, and Se-Yoon Oh. Training deep neural networks with synthetic data for off-road vehicle detection. In *20th International Conference on Control, Automation and Systems*, pages 427–431, 2020.
- [40] Jin Ho Kim, Suk-Hyun Seo, Nguyen-Tien Hai, Bo Mu Cheon, Young Seo Lee, and Jae Wook Jeon. Gateway framework for in-vehicle networks based on CAN, FlexRay, and ethernet. *IEEE Transactions on Vehicular Technology*, 64(10):4472–4486, 2015.
- [41] Dmitrii Kirov, Pierluigi Nuzzo, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. ArchEx: An extensible framework for the exploration of cyber-physical system architectures. In *Proceedings of the 54th Design Automation Conference*, DAC 2017, Austin, TX, June 18–22, 2017.
- [42] Dmitrii Kirov, Pierluigi Nuzzo, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. Optimized selection of wireless network topologies and components via efficient pruning of feasible paths. In *Proceedings of the 55th Design Automation Conference*, DAC 2018, San Francisco, CA, June 24–28, 2018.
- [43] Dmitrii Kirov, Pierluigi Nuzzo, Alberto L. Sangiovanni-Vincentelli, and Roberto Passerone. Efficient encodings for scalable exploration of cyber-physical system architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(1), January 2024.
- [44] Andras Kokuti, Ahmed Hussein, Pablo Marín-Plaza, Arturo de la Escalera, and Fernando García. V2X communications architecture for off-road autonomous vehicles. In *International Conference on Vehicular Electronics and Safety*, pages 69–74, 2017.
- [45] Anne Kozirolek, Heiko Kozirolek, and Ralf Reussner. Peropteryx: automated application of tactics in multi-objective software architecture optimization. QoSA-ISARCS '11, pages 33–42, New York, NY, USA, 2011. Association for Computing Machinery.
- [46] Daniel Kraus, Hristo Ivanov, and Erich Leitgeb. Approach for an optical network design for autonomous vehicles. pages 1–6, 07 2019.

- [47] Daniel Kraus, Hristo Ivanov, and Erich Leitgeb. Approach for an optical network design for autonomous vehicles. In *21st International Conference on Transparent Optical Networks*, 2019.
- [48] Stefan Kugele, Gheorghe Pucea, Ramona Popa, Laurent Dieudonné, and Horst Eckardt. On the deployment problem of embedded systems. In *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEM-OCODE)*, pages 158–167, 2015.
- [49] Chih-Ping Li, Jing Jiang, Wanshi Chen, Tingfang Ji, and John Smee. 5g ultra-reliable and low-latency systems design. In *European Conference on Networks and Communications (EuCNC)*, pages 1–5. IEEE, 2017.
- [50] R. Li, R. Etemaadi, M. T. M. Emmerich, and M. R. V. Chaudron. An evolutionary multiobjective optimization approach to component-based software architecture design. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 432–439, 2011.
- [51] Xiaopan Li, Siqin Chang, and Xiaoxiang Gong. Modeling of a new brake by wire system based on the direct-drive electro-hydraulic brake unit. In *Advanced Information Technology, Electronic and Automation Control Conference*, pages 211–215, 2015.
- [52] Ghyoor Arshad Lodi, Andreas Ott, Sher Ali Cheema, Martin Haardt, and Thomas Freitag. Power line communication in automotive harness on the example of local interconnect network. In *International Symposium on Power Line Communications and its Applications*, pages 212–217, 2016.
- [53] Feng Luo, Bowen Wang, Zhenyu Yang, Ping Zhang, Yifei Ma, Zihao Fang, Mingzhi Wu, and Zhipeng Sun. Design methodology of automotive time-sensitive network system based on omnet++ simulation system. *Sensors*, 22(12), 2022.
- [54] Jonas Maier and Hans-Christian Reuss. Design of zonal e/e architectures in vehicles using a coupled approach of k-means clustering and dijkstra’s algorithm. *Energies*, 16(19), 2023.
- [55] Shane McLean, Emil Hansen, Paul Pop, and Silviu Craciunas. Configuring adas platforms for automotive applications using metaheuristics. *Frontiers in Robotics and AI*, 8, 01 2022.

- [56] Ayhan Mehmed, Moritz Antlanger, and Wilfried Steiner. The monitor as key architecture element for safe self-driving cars. In *50th Annual IEEE-IFIP International Conference on Dependable Systems and Networks-Supplemental Volume*, pages 9–12, 2020.
- [57] P. Meyer, F. Korf, T. Steinbach, and T. C. Schmidt. Simulation of mixed critical in-vehicular networks. *ArXiv*, abs/1808.03081, 2019.
- [58] Philipp Meyer, Franz Korf, Till Steinbach, and Thomas C. Schmidt. Simulation of mixed critical in-vehicular networks. *ArXiv*, abs/1808.03081, 2018.
- [59] J. Miura, M. Ito, and Y. Shirai. A three-level control architecture for autonomous vehicle driving in a dynamic and uncertain traffic environment. In *Proceedings of Conference on Intelligent Transportation Systems*, pages 706–711, 1997.
- [60] Ali Moin, Pierluigi Nuzzo, Alberto L. Sangiovanni-Vincentelli, and Jan M. Rabaey. Optimized design of a human intranet network. In *54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017.
- [61] Ahmed Nasrallah, Akhilesh S. Thyagaturu, Ziyad Alharbi, Cuixiang Wang, Xing Shao, Martin Reisslein, and Hesham Elbakoury. Performance comparison of iee 802.1 tsn time aware shaper (tas) and asynchronous traffic shaper (ats). *IEEE Access*, 7:44165–44181, 2019.
- [62] Rihards Novickis, Aleksandrs Levinskis, Roberts Kadikis, Vitalijs Fescenko, and Kaspars Ozols. Functional architecture for autonomous driving and its implementation. In *17th Biennial Baltic Electronics Conference*, pages 1–6, 2020.
- [63] Pierluigi Nuzzo, Nikunj Bajaj, Michael Masin, Dmitrii Kirov, Roberto Passerone, and Alberto L. Sangiovanni-Vincentelli. Optimized selection of reliable and cost-effective safety-critical system architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2109–2123, 2020.
- [64] Pierluigi Nuzzo, Huan Xu, Necmiye Ozay, John B. Finn, Alberto L. Sangiovanni-Vincentelli, Richard M. Murray, Alexandre Donzé, and Sanjit A. Seshia. A contract-based methodology for aircraft electric power system design. *IEEE Access*, 2:1–25, 2014.
- [65] NXP. Automotive Security: From Standards to Implementation. Accessed: 2024-02-11.

- [66] Official Journal of the European Union. Regulation (EU) 2019/2144 of the european parliament and of the council on type-approval requirements for motor vehicles and their trailers, and systems, components, Nov. 2019.
- [67] Communication over Real-Time Ethernet Group. Core-rg. Accessed: 2024-01-10.
- [68] Maryam Pahlevan, Jonas Schmeck, and Roman Obermaisser. Evaluation of tsn dynamic configuration model for safety-critical applications. In *2019 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 566–571, 2019.
- [69] Roberto Passerone, Daniela Cancila, Michele Albano, Sebti Mouelhi, Sandor Plosz, Erkki Jantunen, Anna Ryabokon, Emine Laarouchi, Csaba Hegedus, and Pal Varga. A methodology for the design of safety-compliant and secure communication of autonomous vehicles. *IEEE Access*, 7:125022–125037, 2019.
- [70] Gaetano Patti and Lucia Lo Bello. Performance assessment of the IEEE 802.1Q in automotive applications. In *International Conference of Electrical and Electronic Technologies for Automotive*, pages 1–6, 2019.
- [71] Q. Pham, F. Fang, V. N. Ha, M. J. Piran, M. Le, L. B. Le, W. Hwang, and Z. Ding. A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art. *IEEE Access*, 8:116974–117017, 2020.
- [72] Paul Pop, Michael Raagaard, Silviu Craciunas, and Wilfried Steiner. Design optimization of cyber-physical distributed systems using iee time-sensitive networks (tsn). *IET Cyber-Physical Systems: Theory and Applications*, 1, 11 2016.
- [73] Tadeus Prastowo, Ayub Shah, Luigi Palopoli, Roberto Passerone, and Giuseppe Piro. Migration-aware optimized resource allocation in B5G edge networks. In *Proceedings of the 19th IEEE Consumer Communications and Networking Conference*, Las Vegas, NV, January 8–11, 2022.
- [74] S. Ramanjaneyulu. Qualitative examination of credit based shaper in IEEE 802.1AVB. *International Journal of Engineering and Advanced Technology (IJEAT)*, 9, Dec. 2019.

- [75] Dominik Reinhardt and Markus Kucera. Domain controlled architecture - a new approach for large scale software integrated automotive systems. pages 221–226, 01 2013.
- [76] Michael Reke, Daniel Peter, Joshua Schulte-Tigges, Stefan Schiffer, Alexander Ferrein, Thomas Walter, and Dominik Matheis. A self-driving car architecture in ROS2. In *International SAUPEC/RobMech/PRASA Conference*, pages 1–6, 2020.
- [77] Soeren Rumpf, Till Steinbach, Franz Korf, and Thomas C. Schmidt. Software stacks for mixed-critical applications: Consolidating ieee 802.1 avb and time-triggered ethernet in next-generation automotive electronics. In *2014 IEEE Fourth International Conference on Consumer Electronics Berlin (ICCE-Berlin)*, pages 14–18, 2014.
- [78] SAE. J1939-71 - surface vehicle recommended pratice, 2012.
- [79] SAE. ISO 26262-1:2018 Road vehicles — Functional safety, 2018.
- [80] SAE. ISO/SAE 21434:2021 Road vehicles — Cybersecurity engineering, 2021.
- [81] H. Saleem, F. Riaz, L. Mostarda, M. A. Niazi, A. Rafiq, and S. Saeed. Steering angle prediction techniques for autonomous ground vehicles: A review. *IEEE Access*, 9:78567–78585, 2021.
- [82] Bernhard Schätz, Sebastian Voss, and Sergey Zverlov. Automating design-space exploration: Optimal deployment of automotive sw-components in an iso26262 context. volume 2015, 07 2015.
- [83] K. Scicluna, C. Spiteri Staines, and R. Raute. Torque feedback for steer-by-wire systems with rotor flux oriented PMSM. In *19th International Conference on Electrical Drives and Power Electronics*, 2017.
- [84] T Selvathai, Jayashree Varadhan, and Swarna Ramesh. Road and off road terrain classification for autonomous ground vehicle. In *International Conference on Information Communication and Embedded Systems*, pages 1–3, 2017.
- [85] McKinsey’s Semiconductors. Getting ready for next-generation e/e architecture with zonal compute. Accessed: 2024-18-11.

- [86] Ayub Shah and Roberto Passerone. Architectural exploration and design for ultra-reliable low-latency indoor robotics systems. In *Proceedings of the IEEE Consumer Communications and Networking Conference*, CCNC 2024, Las Vegas, NV, January 6–9, 2024.
- [87] Mina Shi, Xinhui Liu, Yaowu Shi, Wei Chen, and Qingbo Zhao. Research on the sliding mode based ADRC for hydraulic active suspension of a six-wheel off-road vehicle. In *International Conference on Electronic and Mechanical Engineering and Information Technology*, volume 2, pages 1066–1069, 2011.
- [88] Chau-Chung Song, Wei-Chi Chen, Chen-Fu Feng, and Der-Cherng Liaw. Study of a vehicular drive-by-wire system based on FlexRay protocol. In *Proceedings of SICE Annual Conference*, pages 3618–3624, 2010.
- [89] Till Steinbach, Philipp Meyer, Stefan Buschmann, and Franz Korf. Extending OMNeT++ towards a platform for the design of future in-vehicle network architectures. *CoRR*, abs/1609.05179, 2016.
- [90] Tejas Subramanya, Davit Harutyunyan, and Roberto Riggio. Machine learning-driven service function chain placement and scaling in MEC-enabled 5G networks. *Computer Networks*, 166:106980, 2020.
- [91] Bob Toughton, Tom Galluzzo, Danny Kent, and Carl Crane. Perception and planning architecture for autonomous ground vehicles. *Computer*, 39(12):40–47, 2006.
- [92] S. Tuohy, M. Glavin, C. Hughes, E. Jones, M. Trivedi, and L. Kilmartin. Intra-vehicle networks: A review. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):534–545, 2015.
- [93] UNECE. UN regulation no. 141 - uniform provisions concerning the approval of vehicles with regard to their tyre pressure monitoring systems (tpms), 2021.
- [94] UNECE. UN regulation no. 151 - uniform provisions concerning the approval of motor vehicles with regard to the blind spot information system for the detection of bicycles, 2021.
- [95] UNECE. UN regulation no. 155 - cyber security and cyber security management system, 2021.

- [96] UNECE. UN regulation no. 158 - uniform provisions concerning the approval of devices for reversing motion and motor vehicles with regard to the driver's awareness of vulnerable road users behind vehicles, 2021.
- [97] UNECE. UN regulation no. 159 - uniform provisions concerning the approval of motor vehicles with regard to the moving off information system for the detection of pedestrians and cyclists, 2021.
- [98] Andras Varga. *A Practical Introduction to the OMNeT++ Simulation Framework*, pages 3–51. Springer International Publishing, Cham, 2019.
- [99] B Vinodh Kumar and J Ramesh. Automotive in vehicle network protocols. In *International Conference on Computer Communication and Informatics*, pages 1–5, 2014.
- [100] Kasi Viswanath, Kartikeya Singh, Peng Jiang, P.B. Sujit, and Srikanth Saripalli. OFFSEG: A semantic segmentation framework for off-road driving. In *17th International Conference on Automation Science and Engineering*, pages 354–359, 2021.
- [101] Sebastian Voss, Johannes Eder, and Florian Hölzl. Design space exploration and its visualization in autofocus3. In *Software Engineering*, 2014.
- [102] Gregor Walla, Zaur Molotnikov, Hans-Ulrich Michel, Walter Stechele, Andreas Barthels, and Andreas Herkersdorf. A design space exploration framework for automotive embedded systems and their power management. pages 228–234, 05 2013.
- [103] Jiang Wan, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. Cyber-physical codesign at the functional level for multidomain automotive systems. *IEEE Systems Journal*, 11(4):2949–2959, 2017.
- [104] Jian Wang, Qian Song, Zhibiao Jiang, and Zhimin Zhou. A novel InSAR based off-road positive and negative obstacle detection technique for unmanned ground vehicle. In *IEEE International Geoscience and Remote Sensing Symposium*, pages 1174–1177, 2016.
- [105] Narada Warakagoda, Johann Dirdal, and Erlend Faxvaag. Fusion of LiDAR and camera images in end-to-end deep learning for steering an off-road unmanned

- ground vehicle. In *22th International Conference on Information Fusion*, pages 1–8, 2019.
- [106] Wayne L. Winston. *Operations Research by Wayne L. Winston 4th Edition*. Curt Hinrichs, 10 Davis Drive - Belmont, CA 94002, USA, 2004.
- [107] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.
- [108] Xiuli Yang, Yanhong Huang, Jianqi Shi, and Zongyu Cao. A performance analysis framework of time-triggered ethernet using real-time calculus. *Electronics*, 9(7), 2020.
- [109] Gang Ye, Guangming Xiong, Ning Li, Min Zhu, and Jianwei Gong. Road recognition for a wheeled heavy duty off-road autonomous vehicle. In *International Conference on Vehicular Electronics and Safety*, pages 91–95, 2015.
- [110] De Jong Yeong, John Barry, and Joseph Walsh. A review of multi-sensor fusion system for large heavy vehicles off road in industrial environments. In *31st Irish Signals and Systems Conference*, pages 1–6, 2020.
- [111] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry, and Joseph Walsh. Sensor and sensor fusion technology in autonomous vehicles: A review. *Sensors*, 21(6), 2021.
- [112] Wenhan Zhan, Chunbo Luo, Geyong Min, Chao Wang, Qingxin Zhu, and Hancong Duan. Mobility-aware multi-user offloading optimization for mobile edge computing. *IEEE Trans. on Vehicular Tech.*, 69(3):3341–3356, 2020.
- [113] Lin Zhao, Feng He, Ershuai Li, and Jun Lu. Comparison of time sensitive networking (TSN) and TTEthernet. In *37th Digital Avionics Systems Conference*, pages 1–7, 2018.
- [114] Bowen Zheng, Hengyi Liang, Qi Zhu, Huafeng Yu, and Chung-Wei Lin. Next generation automotive architecture modeling and exploration for autonomous driving. In *Computer Society Annual Symposium on VLSI*, pages 53–58, 2016.