Migrating to Ethernet-Based Centralized Automotive Architectures

Migrating to Ethernet-Based Centralized Automotive Architectures

By

Ryan Kapinski, B.Eng.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements for the Degree

Master of Applied Science

McMaster University

Master of Applied Science (2024)	McMaster University
(Software Engineering)	Hamilton, Ontario

Title:	Migrating to Ethernet-Based Centralized Automotive Architectures
Author:	Ryan Kapinski, B.Eng. (McMaster University)
Supervisor:	Dr. Vera Pantelic, Dr. Alan Wassyng
Number of Pages:	vi, 68

Abstract

Automotive Electrical/Electronics architectures of modern vehicles are currently going through a large shift from distributed to centralized systems. As the number of software features grows and the desire to be user-configurable is becoming ever more important, manufacturers are moving towards flexible centralized platforms. Many new technologies are being used to enable this shift to centralization, such as Automotive Ethernet. Using a workbench constructed with state of the art hardware as a testing platform, an investigation into the process of migrating Electronic Control Unit software from CAN-based distributed architectures into centralized architectures with CAN-FD and Automotive Ethernet as their primary networks was performed. An analysis of how Time Sensitive Networking (TSN) extensions can be used to provide real-time capabilities within automotive Ethernet networks is presented. Further, a partial assurance case is constructed for a system using TSN's Time Aware Shaper to provide time-triggered real-time communication. It is intended for use by system designers to assist in the safety analysis of systems using TSN.

Acknowledgments

I would like to thank my supervisors Dr. Vera Pantelic and Dr. Alan Wassyng for supervising me throughout the process of my thesis.

Thank you Dr. Victor Bandur and Dr. Vera Pantelic for guiding me through research and publishing my first papers, as well as sharing your expert knowledge with me. Your help has been invaluable throughout my graduate studies, and it is extremely appreciated.

Thank you Dr. Mark Lawford for giving me the opportunity to work on an exciting project with state of the art tools, I appreciate all of the experience I have gained throughout my time on this project.

Thank you Matthew Dawson for sharing your extensive knowledge in automotive software development and pretty much everything software related.

Thank you everyone at the McMaster Centre for Software Certification. You have provided a great environment to work in, and I had a pleasure working with all of you.

I would like to thank my committee members Dr. Istvan David and Dr. Neerja Mhaskar for taking part in my committee and giving me very quick and helpful feedback.

Contents

List of Figures

List of Acronyms

TSN Time Sensitive Networking

GSN Goal Structuring Notation

TAS Time Aware Shaper

E/E Electrical/Electronics

ECU Electronic Control Unit

HIL Hardware In-the-Loop

PDU Protocol Data Unit

CAN Controller Area Network

LIN Local Interconnect Network

gPTP Generic Percision Time Protocol

TDMA Time Division Multiplexed Access

Chapter 1

Introduction

The automotive industry is currently experiencing a shift in the design of automotive Electrical/Electronics (E/E) architectures at the time of writing this thesis [Apostu et al. 2019]; [Zhou et al. 2021]. This shift in design is driven by the prospect of providing a more flexible architecture that simplifies the integration of new software features into vehicles through mechanisms such as over-the-air updates [Lee et al. 2016]; [Vetter et al. 2020] and better abstraction of hardware platforms [Mauser, Wagner, and Ziegler 2023]. These new architectures are also based upon networking technologies such as IP over Ethernet, which has the potential to increase bandwidth [Hank et al. 2013], simplify wiring harnesses [Maul, Becker, and Bernhard 2018]; [Jang et al. 2023], and improve plug and play capabilities [Stoll et al. 2021] when compared to more traditional networks such as CAN.

Traditional automotive E/E architectures are comprised of many Electronic Control Unit (ECU)s with functionalities that are focused to a few special purpose tasks [Stähle et al. 2013]; [Staron 2021]. There is usually strong coupling between software and hardware, which increases the effort required to design

a new ECU. ECUs in traditional architectures communicate with each other over a bus network such as CAN. In centralized architectures, the functionality of multiple traditional ECUs is consolidated into more powerful centralized computing platforms [Bandur et al. 2021]. To accomplish this consolidation, computing platforms need to be more powerful and the design of these platforms needs to be more flexible [Zhu et al. 2021]. Another aspect of centralized architectures is the use of Ethernet, which has the potential to provide a more generic solution to networking and enabling the design of service oriented architectures [Gopu, Kavitha, and Joy 2016], providing even more freedom in terms of software reloaction and modification.

1.1 Motivation

When designing the architecture of a complex system such as a vehicle, the practice of modularizing the system into many components with focused functionality is necessary in managing the complexity and physical limitations of such a system. Starting in the early stages of the vehicle's E/E architecture design, the system will be split into logical components each implementing a specific functionality. The traditional method of realizing these components in the physical architecture is to map each functionality to a physical component in the vehicle. Components are interfaced with each other through various networks.

In automotive terms, these discrete components are called ECUs. For each ECU, the development process can be lengthy and involve many different teams to develop. For example, the process for developing an ECU may include a range of activities from designing the physical board, creating a board support

package for the board, configuring the basic software services that supports the application (operating system, communications stacks, hardware signals, etc.), and then designing the application itself. This process will need to be repeated for each ECU in the system. This architecture is beneficial in many different aspects, one of which is the simplification of the design process through being able to reason about each component individually during the design.

To reduce the amount of ECUs required in a system using this distributed architecture OEMs are employing centralized architectures. Moving the functionality of multiple traditional ECUs into a single more powerful controller can simplify the physical architecture, saving on costs associated with bringing up many ECUs. Migrating the functionality of traditional ECUs however does incur its own cost, since they were designed to be deployed in a radically different architecture. This thesis gets its motivation from the need to maximize re-use during the migration, since one of the potential blockers an OEM might face when moving to a centralized architecture is the effort needed to migrate existing assets into a completely new architecture. Additionally, this thesis is motivated by the problem of having a centralized Ethernet network provide real-time communication. This point is critical, because many features in the powertrain domain require real-time communication in order to function safely. Since functional safety is a mandatory property of automotive systems, there is considerable interest in ensuring that these centralized network are capable of providing real-time guarantees.

1.2 Contributions of this Thesis

The contributions of this thesis are in the area of networking within centralized automotive electronics architectures. In particular, this thesis explores topics regarding Ethernet networks in modern centralized automotive systems as automotive Ethernet is currently the primary candidate networking technology within centralized automotive systems that has already seen some adoption in modern cars. Contributions are as follows:

- Several aspects regarding the work needed to migrate existing ECUs designed for legacy decentralized architectures into modern centralized architectures are explored. This topic is very relevant as OEMs need to migrate functionality from existing ECUs into the new architecture in order to reap the benefits of centralized architectures. This is not a trivial task, and the work in this thesis is aimed at determining what the migration entails. The software environment of the ECU will need to change, since the centralization of computing platforms requires deploying many functionalities onto fewer resources, thus requiring mechanisms to properly deploy more features onto single platforms. The changes required are discussed in this thesis, specifically regarding the required changes to ECU networking software, and the considerations of a high bandwidth Ethernet stack. Important differences are pointed out between CAN and Ethernet networks, and various techniques are explored regarding how to migrate from CAN to Ethernet. The safety and reliability of Ethernet in the automotive context is also considered. The mechanisms provided by the IEEE TSN standards are discussed and how they can be used to meet safety and reliability requirements.

- Safety related topics of Ethernet networks in automotive systems have been explored. Ethernet networks in the new centralized architectures will need to facilitate hard real-time communication. Many of these hard real-time requirements are safety-related requirements. Therefore, the network's ability to meet these real-time requirements must be assured in order for the system to be deemed safe. This is particularly important for Ethernet networks because in its basic configuration, Ethernet is best effort meaning that there is no guarantee as to when or even if packets will be delivered; rather, this is dependent on the current state of the network with respect to load, which can be very difficult or impossible to determine. The mechanisms contained in the IEEE TSN standards can be used to prioritize certain traffic and provide some guarantees, but the orchestration of multiple systems, and careful design of the network are needed to assure these timing properties. The contribution of this thesis is to propose an argument structure that aims to assure that a specific Ethernet network will meet its desired timing specification. This argument is presented using Goal Structuring Notation (GSN) as to improve the readability of the argument and explicitly state how the overall goal of meeting the timing specification is decomposed into sub-goals relating to specific aspects of the network design. The specific network that is analyzed in this section is an Ethernet TSN network using the 802.1Qbv mechanisms to isolate real-time communication from other communications in a Time Division Multiplexed Access (TDMA) fashion. The goal of this contribution is to provide designers an argument structure that states the various properties that need to be shown, in the context of TSN mechanisms, in order to assist in the design of a network that can assure

these timing properties. The argument in this thesis is not a complete argument, as it would depend on a full design. However, the argument structure can be expanded upon and tailored to a specific instance of an Ethernet TSN network.

1.3 Thesis Outline

Chapter 2 introduces several topics used throughout this thesis including automotive architecture centralization, in-vehicle networking technologies, and Goal Structuring Notation. Chapter 3 goes over related works. Chapter 4 presents an analysis of migrating legacy ECU software from a decentralization CAN based architecture, into two different centralized architectures based on CAN-FD and Ethernet respectively. Chapter 5 presents a partial assurance case containing an argument structure to show that a particular Ethernet network with TSN is able to achieve bounded latency for a specific data stream, using a scheduled traffic approach. Finally, the final conclusions of this thesis are presented in Chapter 6

Chapter 2

Preliminaries

2.1 Centralized Automotive E/E Architectures

The centralization of automotive architectures can be summarized as the consolidation of features into a smaller amount of more powerful ECUs. Centralized architectures are categorized based on how the features are chosen to be allocated to an ECU, which might be for one of several reasons. In a *Domain Centralized* architecture, features are allocated to an ECU based on being part of a common functional domain within the vehicle. This architecture results in several domain controllers, one for each functional domain in the vehicle, connected by a high bandwidth backbone network to accommodate inter-domain communication. Each domain also has its own lower bandwidth sub-networks which connect the domain controller to the various sensors and actuators belonging to that domain.

Another more radical centralized architecture is the *Zonal* architecture, where features are allocated to ECUs based on location within the vehicle. In this architecture the vehicle is split into several geometric zones, each con-

taining a zonal controller that implements features relating to the sensors and actuators within that zone of the vehicle. Each zone is comprised of the zonal controller and the various sensors and actuators of the zone which are connected to the zonal controller through small zonal networks (CAN, LIN, etc) or directly connected through hardware signals (analog, digital, PWM, etc). The zones are then connected to eachother through a high bandwidth backbone network, similar to the backbone network in domain centralized architectures. One of the main reasons why zonal architectures are a more radical approach compared to domain architectures, is that the communication requirements of the backbone network are higher in a zonal approach due to communication from intra-domain dependencies now being present on the backbone since a functional domain might now be split between multiple zones of the vehicle.

The implementation of centralized architectures as described above is dependant on multiple technological enablers. The first of which is more powerful hardware, which is necessary to have the computing power to run more features simultaneously on a single ECU. Another enabler is the use of virtualization technologies, allowing for the isolation of various operating systems running on a single controller. This isolation is useful for many reasons, but the most important of which is due to the functional safety requirements placed on the design of these components by ISO 26262 [ISO 2011], which requires the different features to have freedom from interference with eachother. Finally, modern networking technologies such as automotive Ethernet and CAN FD provide high bandwidth communication that is necessary due to the increased communication requirements of modern features. In the case of Ethernet, the use of an IP based networking stack can add additional flexibility to the architecture by implementing networking based communication as services that

can be resolved dynamically through various protocols such as SOME/IP.

2.2 Networking in Automotive E/E Architectures

2.2.1 Overview

An important factor in the design of a vehicle's architecture is how signals are transferred between components. Each component is exposed to one or more communication channels and signals are mapped to these channels based on the requirements of the signal and the capabilities of the networking technology of the channel. Signal specific requirements that affect this mapping are typically timing requirements such as latency and jitter. The most common technologies currently used for automotive control signals are Controller Area Network (CAN), Local Interconnect Network (LIN), and FlexRay. These technologies have two common characteristics: they connect nodes together in a bus topology (the pins of all nodes are electrically connected), and they all transfer discrete units of data called Protocol Data Unit (PDU)s. The PDUs are comprised of a payload which contains the actual data being transferred, and a header which is used to identify the contents of the payload.

LIN is a low bandwidth (up to 20kbps) bus that uses a master/slave communication protocol for transmitting and receiving signals to and from a master node. This makes it most suitable for an ECU that controls a small number of sensors/actuators. CAN is a higher bandwidth (up to 1Mbps) bus than LIN, but rather than using a master/slave protocol, it allows all nodes to send to each other and arbitrates transmit access to the bus based on the priority

of the message that all nodes are trying to transmit at any particular time. FlexRay is even higher bandwidth (up to 10Mbps) and contains features such as time multiplexing to improve the real-time capabilities of the network, and redundant channel configurations to increase the fault tolerance. Of these three technologies, only CAN will be of importance.

The increasingly demanding features of modern vehicles is changing the suitability of CAN as the dominant network, resulting in new technologies being proposed to replace it. Two networking technologies have been proposed to address the shortcomings of CAN: CAN FD and Ethernet. CAN FD aims to improve the bandwidth of CAN while maintaining the resilience and simplicity. Ethernet is a drastically different type of network than CAN and provides much greater bandwidth (up to 1Gbps) and flexibility at the cost of greater complexity. CAN, CAN FD, and Ethernet will now be described in more detail.

2.2.2 CAN / CAN FD

A CAN network consists of multiple nodes connected to the same physical interface consisting of two wires. This gives a CAN network a bus topology where only one node can be transmitting at any given time, while all other nodes are receiving the same data being sent by the transmitter. This makes CAN suitable for both one-to-one communication and one-to-many communication. An example of a simple CAN network is shown in Figure 2.1 The data on the bus is serially transmitted at rates typically in the range of 100-500kbps and is sent in the form of discrete PDUs (also known as messages) that contain two sections: a header and payload. The header contains an identifier that

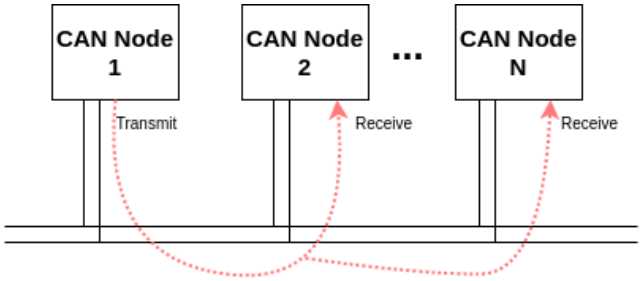

Figure 2.1: CAN nodes are connected to the same physical media in a bus topology. When node 1 transmits a message, all other nodes on the bus will be receiving the same message.

describes what the contents of the payload is as well as the length of the payload (support lengths are 1 - 8 bytes). Using the header identifier, receivers are able to determine if they should process the incoming PDU, and if so, how the data is encoded in to the payload.

In order to avoid collisions on the bus from multiple transmitters, CAN uses a built-in priority based arbitration mechanism that uses the value of the identifier to select which transmitter will succeed in getting access to the bus. The lower the identifier is, the higher the priority of the message. CAN messages are typically sent cyclically using a software scheduler to trigger transmissions, but can also be sent on an event basis.

CAN FD is an extended version of CAN and is very similar in terms of its physical interface, general structure of the PDU, and arbitration mechanism. The main difference with CAN FD is an increased maximum payload size from 8 to 64 bytes, and an increased maximum bitrate of up to 8Mbps.

2.2.3 Automotive Ethernet

Ethernet has become a viable option for in-vehicle networks, as it is more flexible and provides greater bandwidth compared to other networks currently

used such as CAN and FlexRay. Ethernet is not a new technology, being standardized by the IEEE since 1983, however its use in automotive scenarios was limited to non-critical functionality such as diagnostics, due to issues with EMI. However, a new physical layer for Ethernet called 100BASE-T1 was standardized in 2015 [IEEE 2016b] which provides a more EMI compliant wiring interface that allowed the use of 100Mbps Ethernet in a rough environment such as a vehicle. This new physical standard is commonly called Automotive Ethernet. More standard interfaces which fall under Automotive Ethernet followed which are called 1000BASE-T1 and 10BASE-T1 providing a 1Gbps and 10Mbps Ethernet link respectively. For the remainder of this thesis, Automotive Ethernet will be referring to specifically the 100BASE-T1 and 1000BASE-T1 standards. This distinction is important because 10BASE-T1 provides a bus network, where multiple nodes can be connected on the same physical wire. This is different than 100/1000BASE-T1 where connections between nodes are point-to-point, and messages move through a network using specific network devices called switches.

Automotive Ethernet not only provides higher bandwidth networking compared to CAN and FlexRay, but can also provide more flexibility since the IP layer can be used. IP separates the communication requirements from the physical devices, and leads towards a new architectural paradigm called service oriented architectures. In a service oriented architecture ECUs provide and request services during run-time by leveraging the IP stack to provide a dynamically re-configurable architecture. This flexibility can be used to greatly improve the process of dealing with vehicle variants, since services can be added or removed by configuring end nodes to either provide or request that service.

Although Automotive Ethernet provides a fast and flexible networking solution, making an Ethernet network reliable is not as trivial. Without special configuration an Ethernet network operates in best effort, meaning that there is no guarantee that a message sent into a network will reach its destination. This is problematic for vehicle control signals since there needs to be some guarantee that a message will not only reach its destination, but within some timing requirements. In order to make Ethernet a reliable network for control signals, additional mechanisms need to be in place to provide some guarantee that messages will flow through a network with the desired behaviour.

2.2.4 Time Sensitive Networking

One solution for improving the real-time behaviours of automotive Ethernet is to use the mechanisms provided by the Time Sensitive Networking (TSN) group of standards [802.1 n.d.] The TSN standards are a collection of standards created by the TSN task group, part of the IEEE 802 working group which is responsible for standardizing some of the most widely used modern networking technologies (Wi-FI, Ethernet, Network Bridges, etc.). The goal of TSN is to define functionalities for network components that improve the real-time behaviour, such as deterministic latency and priority based packet forwarding, making these networks suitable for applications that require strict timing requirements. The components of TSN can be organized into four main categories: *Synchronization*, *Latency*, *reliability*, and *Resource Management* [Farkas 2018]. To navigate the many components of TSN the task group also standardizes profiles (select components and their configurations) that apply to specific application domains. For example the automotive profile of

TSN is currently, at the time of writing this thesis, covered in [IEEE 2024]. The remainder of this section will cover the specific TSN components that are relevant to the work in this thesis.

2.2.4.1 Synchronization

The synchronization mechanism provided by TSN caters to any use-case that requires the coordination of events in a networked system. In order to coordinate events between several networked components they will need to have a common understanding of the current time which can be achieved by synchronizing the clocks of each component in the network. TSN defines Generic Percision Time Protocol (gPTP) [IEEE 2019], a clock synchronization protocol designed for Ethernet networks that provides a global time base for all participating network nodes to form what is called a domain. gPTP achieves this global time base by distributing the current time of a node called the grandmaster to all nodes in its domain. The protocol defines how to transfer the grandmaster timestamp in such a way that latency is accounted for and each recipient of the timestamp can reconstruct the current upstream time of the grandmaster upon receiving the timestamp. The recipient can then compare the upstream time to its current time and correct it accordingly. gPTP is designed to synchronize clocks in an Ethernet network to within 1us over a maximum of 7 hops from the grandmaster [IEEE 2020]. An example of a gPTP domain is shown in Figure 2.2.

2.2.4.2 The Time Aware Shaper

Network latency is an extremely important factor in real-time systems, where the failure of a message to meet its deadline can result in total failure of the sys-

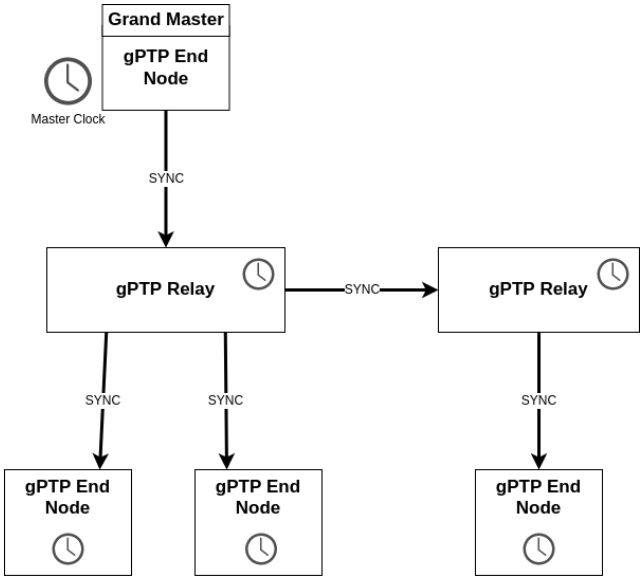

Figure 2.2: All nodes in this gPTP domain are connected through point-to-point Ethernet connections. The grandmaster distributes its current time in SYNC messages to all nodes in the domain, forming a tree. Nodes are either an end instance or a relay instance. Relays have the added ability to forward SYNC messages to either other relays or end nodes.

tem. Due to the packet-forwarding nature of an Ethernet network, switches aggregate streams of packets sharing a common destination from multiple ingress ports into a single egress port. Since only a single packet can be transmitted at a time, this aggregation requires queues on egress ports to deal with multiple packets of a common destination being received at the same time. This queuing can be a source of non-deterministic latency since the current state of the queue is not the same at every point in time, and can vary depending on network traffic. TSN provides mechanisms called traffic shapers for dealing with this queuing latency by enforcing certain behaviours of network devices that result in some control over the state of queues.

One of these traffic shapers is the *Time Aware Shaper (TAS)* [IEEE 2015], which provides TDMA access to the network. It does so by controlling the

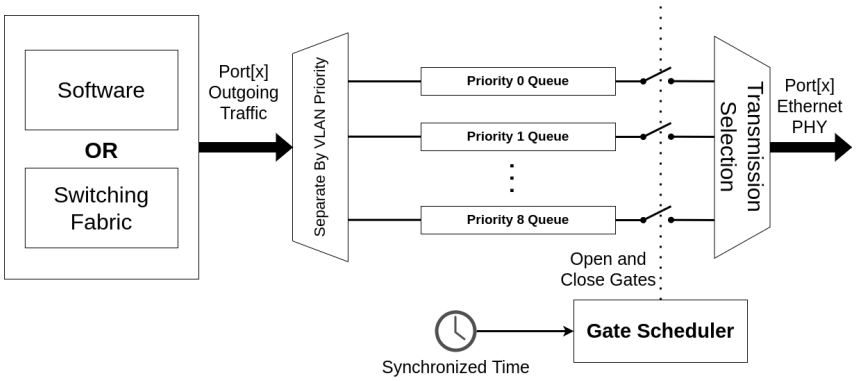

Figure 2.3: The Time-Aware-Shaper is present on the egress path of an Ethernet port. Packets travel from left to right in the figure. Packets originate from either software in the case of end systems, or a switching fabric (packets forwarded from upstream ports) in the case of network switches. Packets are sorted into queues based on VLAN priority. Queues are turned on or off by a gate scheduler based on a synchronized time base. If multiple gates are open at the same time, then the transmission selection component will arbitrate which queue accesses the physical interface.

egress queues of network devices, allocating time slots on a cyclic schedule in which only certain packets can be sent. To achieve this, the egress queue of each port is split into multiple egress queues each with its own priority. These queues are enabled or disabled according to the TAS schedule. If multiple queues are enabled at the same time then arbitration can be done in several ways such as strict priority, or round-robin. The TAS schedule executes cyclically with a period known as the hyperperiod. Within a hyperperiod, there are several time slots where priorities are either enabled or disabled, giving the enabled priorities exclusive access to the network downstream of that port. A high-level block diagram of a TAS is shown in Figure 2.3.

The priority of a packet is determined by the priority code of its VLAN tag, which is an optional field inside of the Ethernet packet header used to logically seperate different types of traffics. The VLAN priority code is a 3-bit field, meaning that any packet can belong to one of 8 priority classes. Thus,

in the TAS there will be 8 egress queues, that are each given access to the downstream network according to the configured schedule. The schedule is executed on a time base that can be synchronized using gPTP allowing all nodes in the network to coordinate the TDMA access of certain priorities across the network.

2.3 Goal Structuring Notation

Goal Structuring Notation (GSN) is a graphical notation used to represent an assurance case in the form of a *GSN diagram*. The syntax and semantics of GSN are standardized in [The Assurance Case Working Group 2021]. Only the core elements of GSN (Core GSN) will be introduced and used in this thesis since they are sufficient for the work contained in chapter 5.

2.3.1 Syntax and Semantics of Core GSN

A GSN diagram is a visual representation of the argument, showing how the premises support the overall conclusion of the argument. The conclusion of the argument is called the *Top-Level Goal*, and the premises that support this top-level goal are called *Goals*. All goals in a diagram, including the top-level goal, are represented by the goal symbol as seen in Figure 2.4. A GSN diagram is read top-down starting with the top-level goal, continuing downwards as each goal is decomposed into sub-goals that together support the claim in the goal.

The relationship between goal and sub-goal is represented by the *SupportedBy* connector symbol as seen in Figure 2.4. The supported by symbol is a directional connector between goals which means that the validity of the source goal is supported by the validity of the destination goal. A goal can

be supported by multiple sub-goals, in which case the validity of the goal is determined by the logical conjunction of the sub-goals.

As goals are decomposed into sub-goals, it will eventually be the case that sub-goals can not be decomposed any further and are instead supported by direct evidence. This direct evidence is represented by the *solution* symbol as seen in Figure 2.4, and is connected to one or many goals that it directly supports. These *solution* elements are the leaf nodes of the diagram and when all paths down from the top-level goal end with solution elements, the argument of the diagram is complete. At this point the top-level goal of the argument is claimed to be true and is supported by evidence in the *solutions* of the diagram, as per the logic represented by the goal decompositions. In addition to goals and solutions, GSN also provides elements to explicitly state the rationale of goals, and provide additional information about the argument structure. These additional elements are called Justifications, Assumptions, and Strategies. The symbols of these elements are shown in Figure 2.4

Justification elements are used to provide rationale for goals and strategies in a GSN diagram. They improve the communication of the argument by explicitly stating why the goal or strategy is used, which also aids the reader to comprehend the argument. *Assumption* elements are used to state any assumptions that are relevant to another element in the diagram, such as assumptions made when stating a goal. *Strategy* elements can be used in the decomposition of goals to provide reasoning for why the sub-goals support the goal being decomposed. An example GSN diagram where each element is shown in 2.5

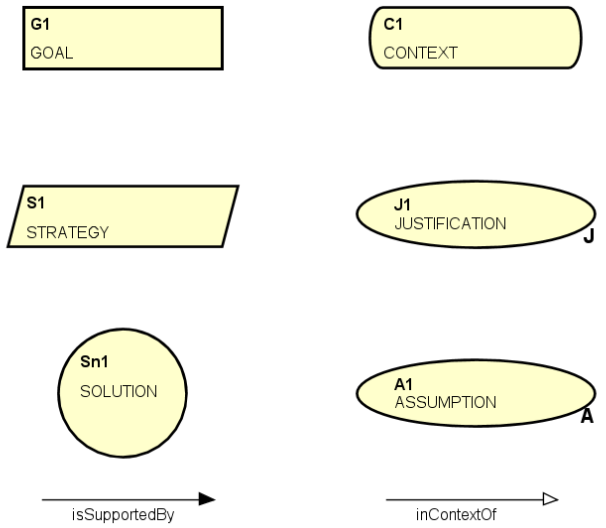

Figure 2.4: The symbols that make up the core syntax of Goal Structuring Notation

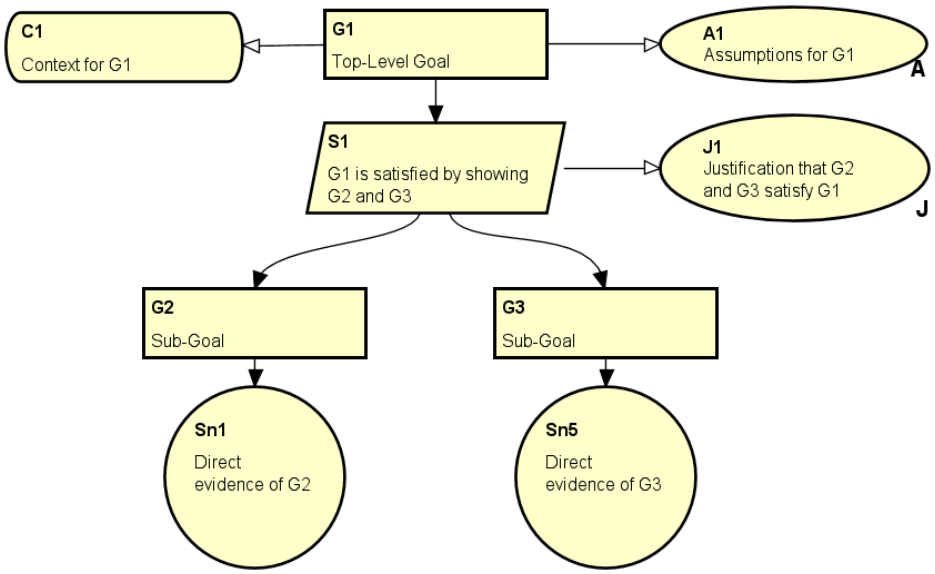

Figure 2.5: This simple GSN diagram shows an example of how each core GSN element is used in a diagram.

Chapter 3

Related Work

The content of this thesis is split between two main topics, both related to the overall theme of automotive architecture centralization. In order to focus the search of related work to these two topics, the scope of relevant work for each topic will be defined in this section.

The first topic of this thesis is the migration of decentralized ECUs into centralized architectures, and an analysis of the networking aspects (migration from CAN based networks to Ethernet and CAN FD based networks). The second topic of this thesis is the assurance of latency in automotive Ethernet networks using Time Sensitive Networking (TSN).

3.1 Migrating From CAN to Ethernet

As the automotive industry adopts Ethernet as the primary backbone network for control communications, the process of effectively migrating control applications from CAN to Ethernet based communications becomes important. Two different categories of strategies exist for this migration.

1. The first involves mixing Ethernet and CAN networks within the same architecture, and using either gateways or encapsulation to transfer CAN PDUs or signals throughout the Ethernet network.

2. The second involves migrating the CAN signals to Ethernet completely, and discarding the presence of CAN PDUs and signals.

This thesis will focus on the first category only, since we emphasize the importance of reusing existing assets which are based on CAN networks. Kern et al [Kern et al. 2011] describe several strategies for encapsulating CAN messages into Ethernet frames. The authors focus on the mapping between CAN messages and Ethernet frames. Their setup includes two different CAN buses that are connected through an Ethernet link, using custom CAN gateways that implement their encapsulation protocols in FPGAs. Using this setup they tested four different strategies; one-to-one mapping between CAN messages and Ethernet frame, buffering CAN messages into frames until a size is reached or a timeout expires, buffering but high priority frame decrease the timeout, and buffering but frames are sent immediately when high priority CAN messages are received.

They found that one-to-one mapping provides the lowest latency, but at the cost of higher overhead since a full Ethernet/IP header is added to the CAN message. They also report that the buffering methods reduce packet overhead, but increase the average latency and jitter of encapsulated CAN messages. The buffering strategies that take into account the CAN message priority decrease the latency/jitter but increase the overhead. This work is related to this thesis because it discusses the performance of different encapsulation strategies for CAN messages over Ethernet. However, topology is not taken into account,

so it is only relevant for cases where messages from a CAN bus are being sent to a specific sender/receiver Ethernet node. This is applicable for cases where CAN buses are being bridged by an Ethernet backbone network.

Berisa et al [Berisa et al. 2023] perform a similar experiment, but add TSN mechanisms to the Ethernet network. The CAN messages are encapsulated into Ethernet frames with stream-reservation class A (using TSN credit based shaper to achieve <2ms latency), and gateways are present on two nodes that bridge two CAN buses through an Ethernet backbone network. Their reason for using the credit based shaper (as opposed to the time aware shaper, which is the focus of this thesis) is that CAN is an event based network, while scheduled traffic with the time aware shaper is determined by a strict offline schedule. This work also focuses on a simple CAN/Ethernet topology, with only two nodes participating in the encapsulation. The encapsulation strategies they tested are FIFO, fixed priority, and one-to-one mapping. The TSN frames are sent periodically, and the number of CAN frames encapsulated per TSN frame was varied, with one-to-one mapping being one CAN frame per TSN frame. They found that encapsulating fewer CAN frames per TSN frame reduces the end-to-end latency of CAN frames, and recommend doing so if the CAN frames have low latency requirements. They also recommend that if the link speed is too slow (ex. 10Mbps) to afford the additional bandwidth overhead of sending fewer CAN frames per TSN frame, then the decision between FIFO and fixed priority should be based on the latency requirements of high priority CAN frames since in the fixed priority scheme high priority CAN frames experience reduced latency.

Zinner et al [Zinner et al. 2011] provide an example implementation of a software based CAN-Ethernet gateway. The gateways is implemented on a

micro-controller with a custom CAN driver, a TCP/IP stack, and some code for packing/unpacking CAN frames into/from Ethernet frames. The gateway strategy for packing CAN frames into Ethernet frames is based on a FIFO buffer. Ethernet frames are sent periodically, and between transmissions the received CAN frames are queued in the FIFO buffer. When an Ethernet frame is received, all encapsulated CAN frames are sent as fast as possible on the CAN side. No performance details of the implementation are reported, only the functionality is tested and reported.

3.2 Assurance of Timing Properties of TSN Networks

Craciunas et. al. [Craciunas et al. 2016] have analyzed the problem of scheduling time-triggered traffic flows in TSN networks with time-aware shapers. They characterized the network scheduling problem by taking into account the topology, message size, message period, link latency, desired message latency, and clock synchronization accuracy into account. Using these parameters and a formal model of how messages will be queued and forwarded through the network switches, they have defined constraints on the network in order to solve for schedules that satisfy their criteria. Assuming that their model is accurate for a particular system, this method will produce network schedules that will guarantee a desired latency. In order for their model to be accurate, various requirements of the networked system need to be guaranteed such as the synchronization accuracy, and the ability for networked devices to send their frames at accurate times relative to the globally synchronized time. The work

in this thesis does not include such a formal model, but does show an assurance case that takes into account the full picture of guaranteed latencies. Therefore, this thesis does not provide an alternative to this work as it does not analyze the problem of schedulability, but rather, it is supplementary as such an analysis would be possible to include in an assurance case similar to the one presented in this thesis.

Maile et. al. [Maile, Hielscher, and German 2020] provide an overview of using network calculus to calculate worst-case latencies for TSN networks using a TAS. Network calculus is an abstract algebra (also called real-time calculus) that can be used to reason about shared resource utilization, in this case for queues in a packet switched network. Network calculus provides the means of analyzing how packets will be serviced in a network, revealing expected worst-case latencies and what size queues will be needed in network switches. This work goes over the application of network calculus for TSN networks, and provides a detailed overview common aspects in the body of research on this topic. Similar to the previous work discussed, the work in this thesis is seen as supplementary to this work. The types of analysis used in this work can be used as a fundamental strategy for assuring latency guarantees, and the assurance case presented in this thesis can be adapted to use such analyses. For example, this analysis requires data streams coming from devices to be maximally bound. The resulting assurance case for such a strategy would thus need to show that these data streams are maximally bound, and reason why it is the case.

Chapter 4

Migrating Legacy ECU Software into Centralized Architectures

A problem that automotive manufacturers face when considering the transition to a centralized architecture is how to utilize already existing assets designed in the context of a distributed architecture. Knowing how to effectively re-use these existing assets can reduce development time by eliminating the need to fully re-implement features from scratch. This chapter provides an analysis of migrating legacy CAN-based ECUs into centralized architectures that are based on either CAN FD or Ethernet networks. The goal of this chapter is to determine how this migration can effectively take place at the networking level, and to identify the design considerations of such a migration.

The results presented in this chapter have already been published in [Bandur et al. 2022]. More precisely, Sections 4.2 - 4.3 were copied verbatim from [Bandur et al. 2022], with minor modifications applied to the content to improve readability and continuity within this thesis. The copied sections contain only contributions to the original work made by the author of this thesis.

This chapter is structured as follows. Section 4.1 will introduce the prototype workbench constructed in order to carry out the migration analysis contained in this chapter. Section 4.2 will discuss various aspects of migrating ECUs from traditional architectures based on CAN networks into centralized architectures based on CAN FD networks. Section 4.3 will discuss various aspects of migrating ECUs from traditional architectures based on CAN networks into centralized architectures based on automotive Ethernet networks. Finally, Section 4.4 gives concluding remarks of the migrations.

4.1 Prototype

We have constructed a domain centralization prototype bench to experiment with migrating legacy powertrain ECUs into a powertrain domain. The major components of this prototype bench are: Hardware In-the-Loop (HIL) simulator, domain controller, domain gateway, and several smart actuators. The HIL simulator provides a simulation of the vehicle itself, including physical signals from within the domain controller (simulating physical devices) and communications from outside of the domain (inter-domain communications). The domain controller is a moderately powerful ECU that implements the high-level controls of the domain. It interfaces with other domain controllers over the inter-domain network, as well as the various smart actuators within the domain. The domain gateway is a physical signal gateway that reads physical signals (Digital, Analog, PWM, etc.) from within the domain, and puts these values on the domain network for the domain controller to access. The smart actuators implement lower-level control of hardware devices (sensors/actuators) and provide an abstracted interface to the component for

higher-level control by the domain controller. Together these components make up a simple domain that we can use to deploy legacy ECUs to create a functional powertrain domain.

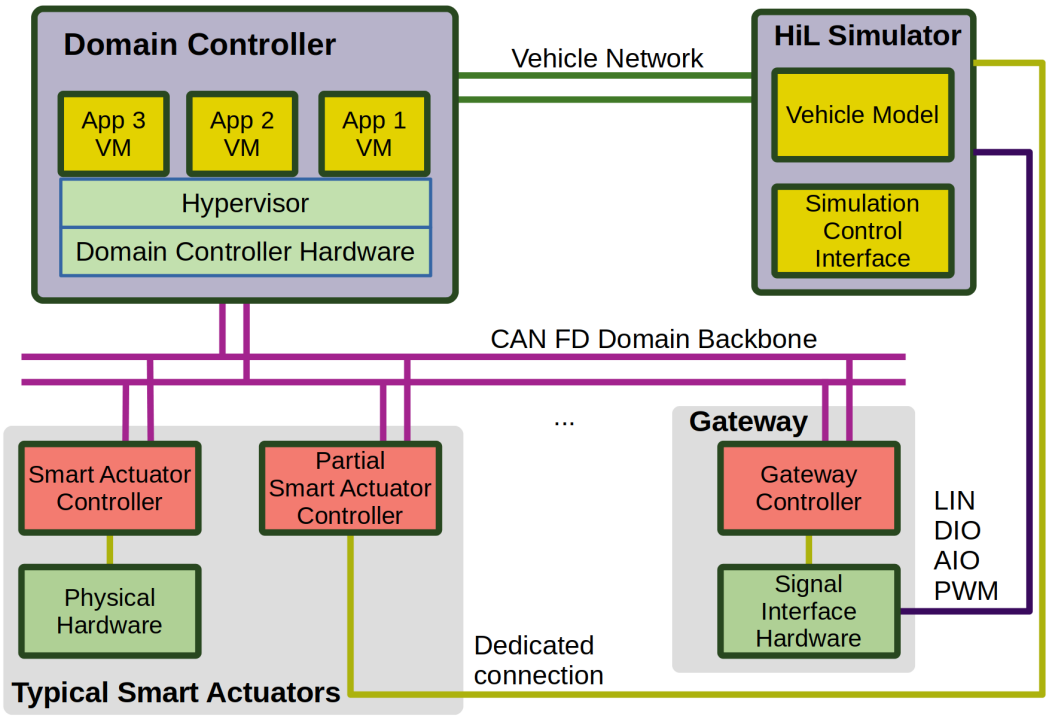

Figure 4.1: The powertrain domain centralization prototype bench with CAN FD as domain network.
Material from: 'Bandur et al, *Aspects of Migrating from Decentralized to Centralized E/E Architectures*. SAE Technical Paper 2022-01-0747. Published 2022, SAE International.' [Bandur et al. 2022] © SAE International

There are two versions of the prototype, as shown in Figures 4.1 and 4.2, which contain CAN FD and Ethernet domain networks respectively. The legacy ECUs related to powertrain functionality were modified to be deployed on this prototype bench, and the various aspects of deploying them were recorded and analyzed. This chapter contains the networking considerations of doing this migration.

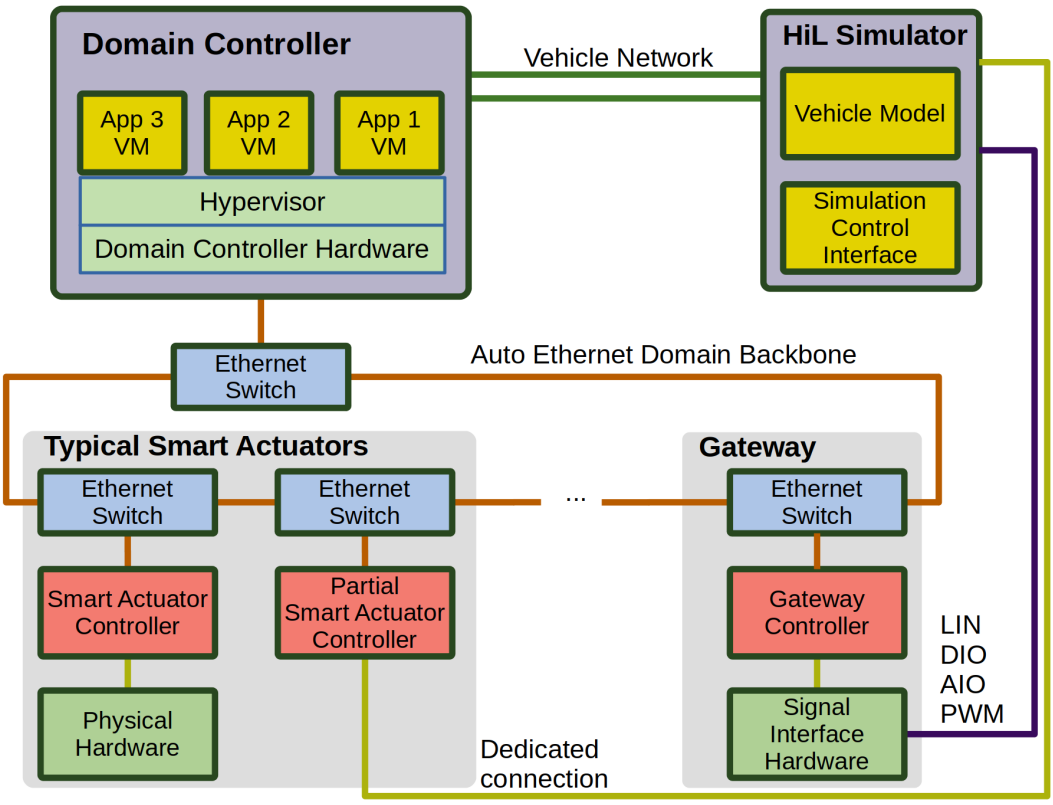

Figure 4.2: The powertrain domain centralization prototype bench with Ethernet as domain network.
Material from: 'Bandur et al, *Aspects of Migrating from Decentralized to Centralized E/E Architectures*. SAE Technical Paper 2022-01-0747. Published 2022, SAE International.' [Bandur et al. 2022] © SAE International

4.2 Migrating ECU Software into CAN FD-based Domain

Our two powertrain domain prototypes differ most significantly in the technology used for the domain communications network. In this section we describe aspects of migrating ECUs into the version of our prototype domain which is based on a CAN FD domain network.

4.2.1 Overview

The first of our powertrain prototypes uses a CAN FD domain network. CAN FD is a "conservative" improvement over CAN, not in the sense of "modest", but rather in the sense that it affords a significant increase in data transmission speed, while remaining identical to the popular CAN bus in almost every other aspect not directly related to the increase in speed. This makes it a viable candidate for simple domain-centralized architectures, because all expertise developed for CAN-based networking can be reused with essentially no additional training of engineers and developers. Tooling also, whether custom-developed or purchased from suppliers, requires minimal modification to support the development of CAN FD networks. Overall, CAN FD retains the same development paradigm, while introducing a significant increase in network communications speed. The relevant considerations in the transition from CAN to CAN FD are discussed in this section.

4.2.2 Major Differences Between CAN and CAN FD

The major shortcomings of CAN are its 8-byte maximum payload length and its maximum bitrate of 1 Mbit/s, which is too slow for many modern requirements. The maximum payload size restricts the amount of data that can be sent per frame, and results in a best-case efficiency of only 58% due to the overhead required to transmit 8 bytes. This low efficiency means that the maximum bitrate of 1 Mbit/s is used mainly to transmit overhead information, limiting the effective payload data rate to about 580 Kbit/s. The CAN FD standard is an extension of CAN that addresses these problems by both increasing the bitrate, and extending the maximum payload size 8-fold to a

64-byte payload, as shown in Figure 4.3. The bitrate is increased through

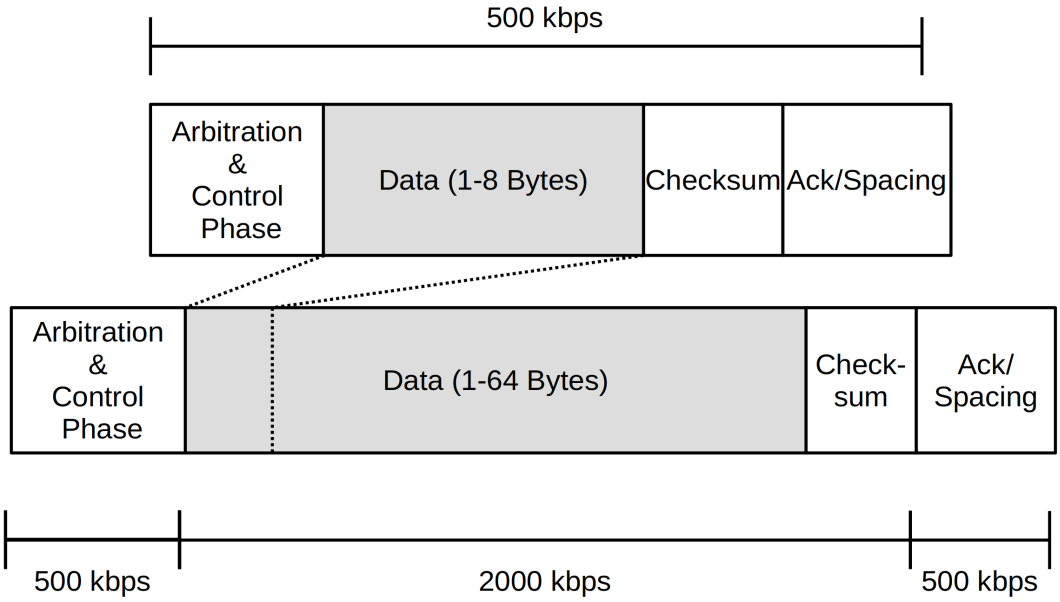

Figure 4.3: The differences in payload size and transmission speed between CAN (top) and CAN FD (bottom). The shaded area represents data payload, while the unshaded area represents the frame overhead.
Material from: 'Bandur et al, *Aspects of Migrating from Decentralized to Centralized E/E Architectures*. SAE Technical Paper 2022-01-0747. Published 2022, SAE International.' [Bandur et al. 2022] © SAE International

enabling a flexible data rate, meaning that the bitrate can increase during the transmission of a message. This flexible rate is required for backward compatibility with CAN, since CAN messages will be arbitrated at a maximum of 1 Mbit/s. After arbitration, the transmitting node that wins is in control and can switch the transmission rate of the bus, which changes the bus to a faster data rate while the node transmits its data. With both the larger payload and the faster data bitrate, the effective speed of payload data transmission is just under 6 Mbit/s [Sinha and Saurabh 2017], an order-of-magnitude increase over CAN.

4.2.3 Migrating ECU Software to CAN FD from CAN

When migrating ECUs from CAN to CAN FD, the simplest solution is to make no changes, since CAN FD is compatible with CAN. However, this is obviously not ideal, since the faster data rate and larger message sizes would not be utilized. In order to benefit from CAN FD, the messages must be repacked with signals, taking into consideration that more signals can be packed into a single message, and that messages can be transmitted at a faster rate. To reduce the number of messages on the bus and maximize efficiency, messages should be designed with large payloads. To leverage CAN FD's potential for a more efficient bus and to maximize data throughput, messages can be designed with large payloads. This may require packing signals into CAN FD messages with slightly different transmit frequency and priority attributes than their respective CAN messages. The analysis of acceptability of such a migration is not always trivial, and may require analysis of the resultant network. The repacking is done during network planning stages of development, and will typically involve tool support to plan and test the resulting network. Once the new network is planned, the result is a network description file (*e.g.* Vector's DBC format) that defines which signals are packed into which messages, and how they are arranged. The structure of the file is very similar between CAN and CAN FD, containing additional attributes specific to CAN FD.

Figure 4.4 shows how the network description file is incorporated into the ECU software. On the ECU, there is a collection of software modules that can be referred to as the CAN stack. The CAN stack interfaces with the ECU application by exchanging signals to and from the application. The application is not aware of where it is receiving from, or where it is sending signals to. It

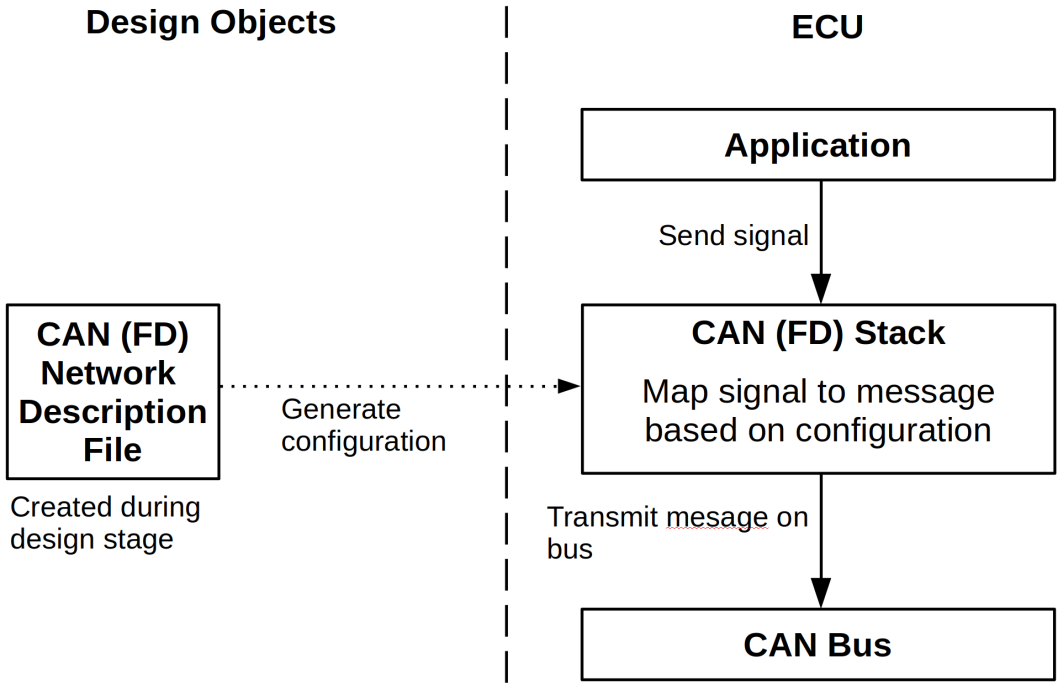

Figure 4.4: Configuration of CAN software stack.
Material from: 'Bandur et al, *Aspects of Migrating from Decentralized to Centralized E/E Architectures*. SAE Technical Paper 2022-01-0747. Published 2022, SAE International.' [Bandur et al. 2022] © SAE International

is only aware of the flow of signals. The CAN stack is responsible for mapping every signal to a CAN message. This separation makes the migration of ECUs to other networks much easier, since the application itself does not need to change. Only the CAN stack itself will potentially need to be modified to add CAN FD support if it does not already exist. The extent of this modification depends on the complexity of the CAN stack.

For example, in our powertrain prototype, the CAN stacks are custom and accompanied by custom tooling to generate configurations from Vector DBC network description files. The signals from the original CAN messages were migrated into CAN FD messages such that the new message had the same transmitting node, and similar transmit frequency and priority. The custom

tooling then needed to be modified to read CAN FD specific attributes in the network description files, and the generated configuration format was modified to include the additional CAN FD attributes. Finally, the CAN stack itself needed to be modified to read the new generated configuration attributes, and take the appropriate action given the configuration. If a purchased supplier stack is used, the supplier will typically offer configuration generation tools that are compatible with CAN FD buses, as long as the target platform has CAN FD support.

Overall, the transition from CAN to CAN FD is fairly straightforward. True to the design intent of CAN FD, minimal modification is required to existing software. The transition focuses effort on the optimization problem posed by the need to change the message structures in order to make full use of the increase in effective data rate. Methods and tools exist for defining and solving such optimization problems [Sandstrom, Norstom, and Ahlmark 2000]; [Bordoloi and Samii 2014] that can be integrated into an automated transition mechanism. We have not yet attempted this in our work. As the CAN FD bus nears full load, optimal message packing becomes increasingly important, and such a mechanism can be greatly beneficial in the context of product lines, if the network design varies at the signal level.

4.3 Migrating ECUs Software into Automotive Ethernet-based Domain

In this section we describe aspects of migrating ECUs into the version of our prototype domain which is based on an automotive Ethernet domain network.

Unlike the transition to CAN FD, this transition requires a lot more thought and work, and leaves a lot of room for experimentation, as the technologies are fundamentally different. Furthermore, there is not yet an established corpus of best practices when designing an automotive network based on Ethernet, making it necessary to iteratively experiment with such E/E architectures to reveal pitfalls.

4.3.1 Overview

From many points of view, automotive Ethernet is the most promising vehicle domain networking technology currently available [Bandur et al. 2021]. The transition to an Ethernet-based domain network presents many options, as the technology is inherently more complex. As an example, as mentioned above, automotive Ethernet provides orders-of-magnitude increases in data rate and maximum payload size. Taken together, these play a vital role in the optimal usage of the communications medium with respect to task data throughput and latency. The problem of re-allocating signals to Ethernet frames becomes even more complex than in the CAN FD case in this respect.

The industry has decades of experience developing optimal CAN-based E/E networks for several dozens of ECUs in the demanding environment of a modern vehicle. However, Ethernet has been used far less in the automotive domain. Therefore, there is far less knowledge, experience and lessons learned to draw upon. Since the AUTOSAR standards included Ethernet communications, suppliers have been distributing Ethernet/IP (TCP and UDP) software stacks as well as providing the tooling to configure the stacks. These stacks typically support signal/PDU-based communications over Ethernet/IP. How-

ever, there is a large focus on the use of Ethernet to promote Service Oriented Architectures in automotive systems using AUTOSAR's SOME/IP middleware [AUTOSAR Consortium 2020]. In this paper we focus on signal-based communications over Ethernet. The field is currently experimenting with this new technology and testing its limits, but it will be a long time before a body of best-practices is established. In this section we report on our own hurdles, findings and insights related to the problem of migrating a legacy E/E network to a centralized one based on automotive Ethernet.

4.3.2 Differences Between Ethernet and CAN (FD)

When looking at Ethernet, we are specifically considering switched Ethernet networks, where all connections are point-to-point, as opposed to networks where all nodes share the same transmission medium[1]. The unit of traffic in an Ethernet network is called a packet, similar to messages in a CAN (FD) network. The most apparent difference between CAN (FD) and Ethernet is that the transmission of Ethernet packets is based on forwarding between nodes called switches, while in a CAN (FD) network all nodes share the same physical bus, where all messages are broadcast to every node on the bus. Ethernet's switched network topology opens up many possibilities for increasing the amount of data going through the network, since multiple nodes can communicate with each other at the same time. In addition to the more flexible topology, the maximum data rate of Ethernet is much higher than that of CAN (FD). The data rate of Ethernet depends on which standard is used at

[1]Nodes on an Ethernet network can also be connected together by sharing the same medium, through the use of hubs, which are passive aggregators of the nodes' communication lines. But in our work we use switches, which are sophisticated active multiplexers that give each node its own dedicated connection.

the physical layer, the standard defining the electrical signals and wiring of the network medium. Currently, the two physical layer standards for switched automotive Ethernet are 100BASE-T1 and 1000BASE-T1 [IEEE 2022], which can transmit at speeds of 100Mbps and 1Gbps, respectively. Specifications for higher rates are being developed. These standards differ from regular, *traditional* switched Ethernet (*e.g.*, home networks) physical layers in that they are designed specifically to be compliant for automotive use (primarily with respect to electromagnetic compatibility).

However, with the flexibility and performance of Ethernet comes increased complexity. In order for an Ethernet packet to get from one node to another, it is required to be actively forwarded through switches. The traditional Ethernet layout consisting of nodes and switches is only capable of sustaining "best-effort" traffic: traffic that is suitable only for applications that can tolerate packet loss and nondeterministic latencies. Traditional Ethernet networking typically relies on higher level protocols such as Transmission Control Protocol (TCP) to ensure that streams of packets are received intact, by retransmitting any packets that may be dropped in the network. Again, in many applications where timely and fresh information is what is expected at the receiving node, this approach is not acceptable, and these high-level protocols can not be used. The alternative is for the devices in the network (including switches) to be compliant with other standards; standards that allow real-time communications to take place on an Ethernet network, in addition to best-effort. This requires certain mechanisms to be in place, which will be discussed later in this section.

4.3.3 Migrating Signals to Ethernet Packets from CAN

CAN communication is signal-oriented, such that individual signals are grouped and mapped into CAN messages in a static format. Signals are grouped into CAN messages based on several attributes, such as signals having the same sending node, receiving node, priority, and rate of transmission (the rate of transmission of the signal by the node, not the data rate of the network carrying the signal). When migrating to Ethernet from an existing CAN network, one option is to redesign the network at the signal level and discard the previous CAN network design. This process is similar to designing a CAN network, with signals being grouped into packets based on their attributes. Obviously, due to the fundamental differences between CAN and Ethernet, signal grouping into Ethernet packets will yield a different format than that of a CAN message. The two most significant differences that affect the signal grouping are the larger size of Ethernet packets, and the fact that Ethernet is a switched network as opposed to a broadcast-based bus. These two aspects are tightly related. If a single packet of maximum size is packed to contain signals relevant to several receiving nodes, then the time of transmission of the single packet must be taken into consideration when deciding how the packet is made to reach each of the relevant nodes, via the unicast or multicast methods, as we discuss later. If, on the other hand, the packets are kept small, then the efficiency of packet transmission decreases due to the reduction in packet payload size, as discussed ealier.

In addition to the work required to solve the problem of optimal grouping of signals into Ethernet packets, the ECU software needs to be reconfigured to send these signals over Ethernet according to the new mapping of signals

to Ethernet packets. This typically requires the integration of supplier software libraries to operate Ethernet hardware and, optionally, transport protocol software to manage the transport of packets on the network, such as UDP/IP. This additional software can be referred to as an Ethernet stack.

The effort required to migrate a CAN network to Ethernet by mapping the signals to new Ethernet packets depends on the current design of the ECU software. The software and hardware of an existing ECU on the network may require extensive modification to provide Ethernet functionality. In cases where this migration requires extensive effort, a simpler solution is to encapsulate CAN messages within Ethernet packets directly. This treats CAN messages (payload and control data such as message ID included) as the data unit, rather than placing signals directly within Ethernet packets. Since Ethernet payloads are much larger than CAN message payloads, multiple CAN messages can fit inside Ethernet packets. One standard that defines such a format for packing the CAN messages into Ethernet packets is IEEE 1722-2016 [IEEE 2016a]. This standard is part of the IEEE Time Sensitive Networking group of standards (discussed in detail later in this section) and describes a transport protocol over Ethernet for time sensitive data streams. It describes data formats for Audio/Visual data, as well as a type called AVTP Control Format (ACF). ACF defines a format for encapsulating several CAN messages within Ethernet frames, which can be used to transport CAN messages throughout an Ethernet network.

Encapsulating CAN messages into Ethernet packets is typically done when CAN networks are present alongside Ethernet networks and communication needs to take place across the two networks. To connect the two different networks, a *gateway* encapsulates the CAN messages into Ethernet packets

and vice versa. Rather than having two separate functional networks, we are specifically interested in *replacing* a CAN network with an Ethernet network by inserting gateways (as either an external device or as a software module within an ECU) such that no two devices will communicate over a physical CAN bus. Encapsulation allows ECU software (and hardware in the case of an external device performing encapsulation) to be less aware or unaware of the network change. However, the mixing of CAN-like communication over an Ethernet network leads to a design optimization problem of how to pack CAN messages into Ethernet packets in order to have the most efficient use of the network as well as to avoid unnecessary congestion of the network. The problem of optimizing the encapsulation of CAN messages into Ethernet packets in an automotive setting has been analyzed previously by Kern *et al.* [Kern, Streichert, and Teich 2011] where unicast routing was used, meaning every packet has a single destination.

Since CAN is a broadcast-based bus network, whenever a node transmits, all other nodes see the transmission. This allows multiple receivers to receive a message from a single transmission. In addition to this, the transmitting node on a CAN bus does not have to route messages; the destination is determined by the message ID and receivers will listen to the message if they are configured to do so. In a switched Ethernet network, packets need to be forwarded through network switches to reach their destination. The forwarding is done using MAC addresses; each end node on the Ethernet network has a specific MAC addresses and the switches know where they need to forward packets in order to reach the end node with a specific MAC address.

Depending on the destination MAC address present in an Ethernet packet, a switch can take one of several actions when forwarding the packet. A packet

can be transmitted using the *unicast*, *multicast*, or *broadcast* schemes. A unicast packet will be transmitted to a single end node, multicast packets will be sent to a specific group of end nodes, and broadcast packets will be sent to all end nodes. Multicast destination nodes are determined by configuring multicast groups in the switches. Taking different network routing schemes into account can add complexity to the problem of optimizing the CAN message to Ethernet packet mapping. For instance, unicast requires the sending node to send the packet once for each recipient of messages or signals contained therein. This multiplies the amount of time spent by that node sending on the network. On the other hand, taking the multicast approach requires simultaneous optimal definition of multicast recipient groups as well as allocation of messages to Ethernet packets, but reduces the node's time spent transmitting. A broadcast approach would simplify the routing problem of packets, since it mimics the broadcast nature of CAN, however it would greatly increase the network load as every node will need to process every packet making it an obviously not optimal solution. It becomes clear how the many options provided by Ethernet lead to complex optimization problems. Of course, finding optimal solutions to these problems yields benefits in terms of scalability and maintainability.

4.3.4 Architectural Options for Encapsulating CAN Messages into Ethernet Networks

When migrating legacy CAN-based devices to an Ethernet network by encapsulating CAN messages in Ethernet packets, the architecture of the solution will depend on how the device is being integrated into the network. In the case

of an ECU being migrated to a virtual machine on a domain controller, the encapsulation of CAN messages can be performed by the ECU VM software, or the encapsulation can be provided as a service from a separate device VM, as shown in Figure 4.5. If the encapsulation is provided by the device VM, the changes required will shift from the ECU software to a separate, reusable solution provided as a service. Within the ECU software, the CAN driver is designed such that the ECU uses its original interface for sending and receiving CAN messages, and when the device VM receives the CAN messages, it provides gateway and routing functionality that encapsulates the CAN messages in Ethernet packets, sending them according to an optimized packing and routing strategy.

If the device containing a CAN-based legacy software component is not within a virtualized environment, but is rather to be integrated as previously designed onto the Ethernet network, the device either needs to be extended to add encapsulation capabilities in software, or an external gateway can be used so that the software component is not aware of the encapsulation. The former is not a simple solution, and will first of all require that the device have Ethernet capabilities. This is not realistic, as the legacy device would not have been designed initially with this in mind. Instead, an external solution can be used to gateway the CAN messages to Ethernet in a more traditional sense. This can be an external, dedicated device, or it can be incorporated in the design of the device as an additional on-board component. The current external solution that we are investigating is a hardware-based IEEE 1722-2016 encapsulation device, in which a dedicated external device has both CAN and Ethernet transceivers. It can connect to a CAN bus (or the individual component), pack the messages directly into Ethernet packets according to its configured

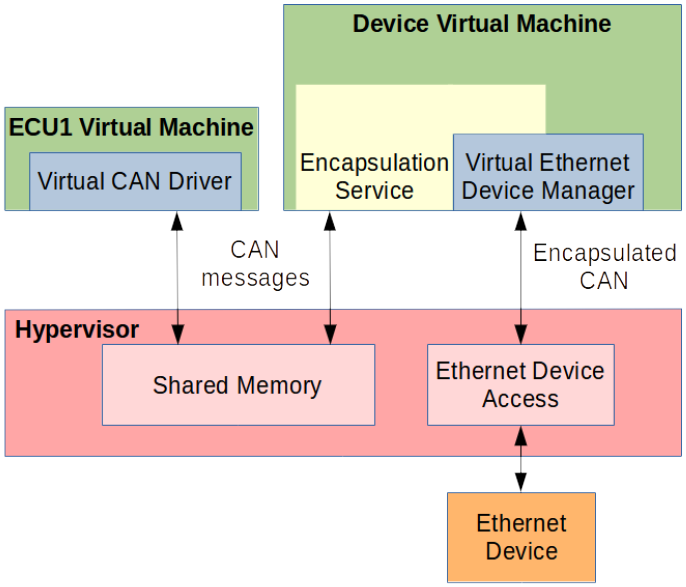

Figure 4.5: Software networking stack of a hypervisor based domain controller. A CAN over Ethernet encapsulation service is provided by seperate device VM. From the ECU VM perspective, it is still using a CAN interface. The virtual CAN driver uses hypervisor provided shared memory to send and receive CAN messages. A Device Virtual Machine processes CAN messages through the shared memory interface and encapsulatse/decapsulates the CAN messages into Ethernet packets sent and received on a physical Ethernet device that it controls.

parameters, and thus make CAN devices available on the Ethernet network. This solution is of interest because it can greatly simplify the migration process. The device only requires a configuration which dictates how CAN messages are packed into Ethernet packets. It is then placed between the two networks, where it performs the encapsulation transparently. Additionally, since the device manages the encapsulation in hardware, there is minimal added latency, as opposed to a software solution where the encapsulation procedure adds appreciable latency.

4.3.5 Ethernet Network: Real-Time and Safety Requirements

In this section, we will be considering support for hard real-time and safety-critical Ethernet communication within the Ethernet-based prototype domain. Mechanisms provided by the IEEE Time Sensitive Networking (TSN) group of standards [IEEE 2017c] will be used to satisfy our network requirements. In particular, we are interested in synchronization, guaranteed low latency, redundancy, and traffic isolation.

Synchronization: The synchronization of time across all network devices (end nodes and switches included) is important for deterministic networking, and is required for the other mechanisms that we are using for deterministic latency. The TSN standard providing time synchronization is 802.1AS [IEEE 2019], which defines generalized Precision Time Protocol (gPTP), based on Precision Time Protocol (PTP) defined in IEEE Std 1588 [IEEE 2008]. gPTP describes a protocol for time synchronization between a grandmaster clock (provided by a device in the network) and all other nodes in what is called a gPTP domain, consisting of a subset of nodes in the network. While the standard defines an algorithm for selecting a grandmaster dynamically (Best Master Clock Algorithm), due to the static nature of this automotive power-train network, we set a specific node as the grandmaster to prevent the need for executing the selection algorithm. The implementation of gPTP in a network requires a PTP instance to be present at every node. The PTP instance carries out the synchronization protocol, and requires PTP timestamping hardware support for accurate synchronization. The PTP instances in our experimen-

tal setup are implemented as a software stack on every node (switches, the domain controller, and smart actuators). This requires a gPTP stack to be supplied or developed for each node in the gPTP domain. In the case of our network switches, there is an internal ARM Cortex-M7 core that manages the switch and executes the gPTP protocol stack. The global clock is shared by all nodes in the PTP domain allowing synchronization of events throughout the participating nodes in the network. It should be noted that a network can contain multiple gPTP domains for different areas of synchronization; however, we are only using a single gPTP domain. Devices without a PTP instance are not part of the PTP domain; therefore, they are not aware of any time synchronization.

Latency: The latency mechanisms in TSN provide means of controlling packet forwarding throughout the network so that other traffic on the network does not interfere with time-sensitive packets. The TSN standard that we are implementing for guaranteed transmission latency is 802.1Qbv [IEEE 2015] which uses Time Division Multiplex Access (TDMA) for scheduling several classes of traffic. When multiple packets going to the same destination enter a switch at the same time, the switch queues the packets and sends them out as fast as possible. When all packets are on the same queue, which occurs in normal operation without assigned traffic classes, this can be classified as best-effort traffic. This congestion of packets causes latency in the network, and needs to be prevented in time-sensitive networks. To deal with this, packets are given a traffic class specification (limited to 8 classes) that are queued separately in switches. There are many TSN mechanisms for handling these different queues. 802.1Qbv gives network access to configured sets of traffic

classes according to a cyclic schedule based on the global gPTP time. For a scheduled time window, the specified time-sensitive traffic classes will have dedicated access to the network with no interference from other less time-sensitive traffic classes. If properly scheduled, the packet latency can be bounded by the propagation time of a packet through the network. The implementation of time scheduled traffic in a TSN network requires the implementation of gPTP for a global clock. Next, a schedule of time-sensitive traffic classes is created that will satisfy the requirements of the time-sensitive information within the Ethernet packets. Finally, the schedule is converted to a format that can be loaded to the switch for configuration. The described process is used for static network configurations. If the traffic requirements are dynamic, then TSN standards defining dynamic network configuration can be used. In addition to 802.1Qbv, the TSN standard 802.1Qci [IEEE 2017b] can be used to prevent babbling-idiot failures in which a network device fails in such a way that it transmits at a rate that congests network traffic. 802.1Qci employs per-stream filtering and policing, which can be used to limit traffic and reserve bandwidth for specific high priority streams of network traffic. If a switch detects that a particular node is transmitting too much, it will detect the abnormal rate and drop packets in order to prevent congestion. Setting up static filtering and policing policies in a network involves first determining policies based on network traffic requirements, and then simply configuring the switches in the network either statically, or during run-time based on operating conditions.

Redundancy: In order to improve the reliability and safety of an Ethernet network, TSN provides a standard for redundant communication. The relevant TSN standard is 802.1CB Frame Replication and Elimination [IEEE 2017a]

that describes a mechanism in which network switches will duplicate packets and send them to their destination across redundant paths. The mechanism also ensures elimination of redundant packets at switches and end devices. This mechanism is only implemented in the switches via switch configurations. The switches are configured to recognize packets that need to be duplicated and to send the duplicates along appropriate redundant paths to reach their destination. The effectiveness of the redundant paths depends on the network topology. For example, our network has a ring topology, meaning that all devices are connected via switches in a circle. Each switch has three connections; one going to the device itself, and two going to adjacent switches. This ensures that whenever nodes communicate there are redundant paths for packets to travel. This topology, along with switches configured to duplicate/eliminate packets, ensures protection against any single link failure, but also results in the number of packets in the network doubling.

Traffic Isolation: In order to guarantee absence of cascading failures on the network, it is necessary to separate different types of traffic such that misidentification of traffic does not lead to interference between traffic streams. This is achieved in TSN Ethernet using the concept of VLAN tagging to identify distinct virtual networks and prevent interference between them at the hardware level. TSN standard 802.1Q [IEEE 2018] defines the mechanism.

In our experimental setup, only a static TSN configuration is used as we do not expect the resource requirements of the network to change dynamically during runtime. As a result, our network does not make use of dynamic network configuration. However, we do see potential use of dynamic network reconfiguration in our application due to e.g. link failures. This will be a

future topic that we will examine.

4.4 Conclusion

When migrating automotive architectures towards a more centralized approach it is important to ensure that existing assets are re-used effectively so that additional efforts are not required to re-implement features from scratch. In this chapter, the migration of legacy powertrain ECUs from a decentralized platform into a centralized domain is explored on a domain centralization prototype bench. The legacy ECUs are migrated into two domain variants that differ by networking technology; one with CAN FD and one with automotive Ethernet. In each case the strategies and technologies used to perform the migration are discussed, as well as implications of moving from one networking technology to another. It was found that transitioning from CAN to CAN FD is a relatively simple migration since the technologies are so similar, with the main difference being that CAN signals can be condensed into fewer messages due to the faster data rate, and larger PDU size. The major software modification is that network descriptions and CAN driver configurations will need to be updated according to the new architecture. The transition to Ethernet is a more radical change, due to the point-to-point connection topology, the and the additional protocols that are encapsulated within a typical packet. The software on the ECU will change significantly, with additional software stacks that deal with messages in quite different ways than CAN. The transmission speed is also significantly increased, which improves bandwidth, but also increases processing requirements in order to process messages at the higher data rate. Finally, the nature of basic Ethernet is not ideal for real-time communi-

cations, since the switching of packets is dynamic and packets can be dropped or queued for non-deterministic amounts of time. The TSN standards are shown to have mechanisms that can be implemented on ECUs and in network hardware to improve the real-time capabilities and reliability of automotive Ethernet. These mechanisms include time-synchronization, priority based isolation of bandwidth via TDMA, message redundancy, and isolation of network streams using VLANs.

Chapter 5

Assuring Real-Time Communication with Automotive Ethernet and TSN

The analysis of real-time properties of networks in cyber-physical systems has been widely studied [Geyer and Carle 2016]. Further, work exists that leverages assurance cases for reasoning about software timing properties (e.g., [Graydon and Bate 2014]). However, to the best of the authors' knowledge, structured arguments such as assurances cases have not yet been applied to TSN networks. Assurance cases (ACs) explicitly present reasoning that a system satisfies a property. In this paper, we present the concept of using ACs for assuring timing requirements of systems containing automotive TSN Ethernet networks. We envision this work to be used to guide developers in understanding which mechanisms should be employed to achieve timing requirements in systems relying on TSN mechanisms, as well as what evidence needs to be provided to assure the timing properties of such systems. Also, it can be leveraged as part

of larger assurance cases assuring properties of systems to which the networks belong. For example, we expect the assurance case to be part of safety cases of modern automotive systems employing automotive Ethernet.

The results presented in this chapter have been already published in [Kapinski et al. 2023]. More precisely, Sections 5.1 - 5.2 were copied verbatim from [Kapinski et al. 2023], with minor modifications applied to the content to improve readability and continuity within this thesis. The copied sections contain only contributions to the original work made by the author of this thesis.

This chapter is structured as follows. Section 5.1 will provide background on the TSN mechanisms used in this Assurance Case. Section 5.2 will go over the construction of the Assurance Case. Finally, Section 5.3 gives some concluding remarks of the final Assurance Case.

5.1 TSN Ethernet

The system to be analyzed in this work is an automotive TSN Ethernet network used for real-time communication. A simplified form is shown in Fig. 5.1. The network is made up of only *end nodes* and *switches*. End nodes are devices with a single port that can source or sink messages in the form of Ethernet packets. Switches are devices with multiple ports that forward packets from one port to another depending on the destination MAC (Media Access Control) address of the packet, and a forwarding table that associates MAC addresses with outgoing ports. Each port can be connected to exactly one other port, and each port has a unique MAC address. While the use of multiple switches depicted here may seem unnecessary, a safety-critical application would use such switch chaining due to communications path redundancy requirements [Bandur et al.

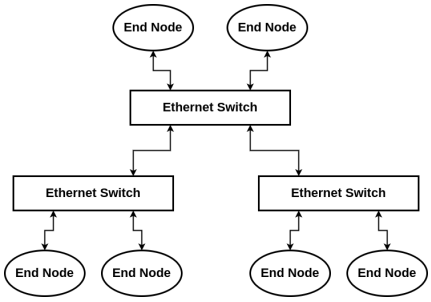

Figure 5.1: A simplified block diagram representing a switched Ethernet Network.
Material from: 'Kapinski et al, *Assurance Cases for Timing Properties of Automotive TSN Networks*. Computer Safety, Reliability, and Security. SAFECOMP 2023 Workshops. SAFECOMP 2023. Lecture Notes in Computer Science, vol 14182. Published 2023, Springer, Cham.' [Kapinski et al. 2023]

2021].

In TSN, the *Time Aware Shaper (TAS)* [IEEE 2015] is based on the fact that Ethernet devices can queue outgoing packets separately based on information in their headers. Each egress (outgoing) port of every Ethernet device in a TSN network has eight queues. Flows in the network (unidirectional streams of Ethernet packets between end nodes) are organized into traffic classes that are used to isolate flows into separate queues based on their requirements. Whenever an Ethernet device's egress port is available for transmission, one of its eight queues is selected, a packet is taken from it and sent out. The TAS employs the traditional TDMA (Time-Division Multiple Access) scheme by enabling and disabling the egress queues according to a schedule. This schedule executes periodically with the schedule *hyperperiod*, and is composed of smaller time slots that define the queue states (enabled or disabled) at a specific phase and duration relative to the hyperperiod. The schedule is executed relative to a clock that is assumed to be synchronized with all other devices in the network, allowing the network as a whole to orchestrate desig-

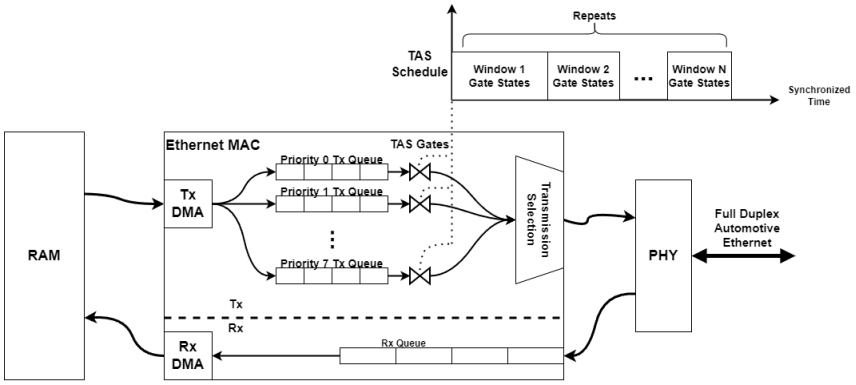

Figure 5.2: Block diagram of a typical Ethernet MAC with a TAS. Rx and Tx paths of packets between system RAM and physical Ethernet media are denoted by arrows.
Material from: 'Kapinski et al, *Assurance Cases for Timing Properties of Automotive TSN Networks*. Computer Safety, Reliability, and Security. SAFECOMP 2023 Workshops. SAFECOMP 2023. Lecture Notes in Computer Science, vol 14182. Published 2023, Springer, Cham.' [Kapinski et al. 2023]

nated times for specific traffic classes to flow. A simplified block diagram of a typical Ethernet interface with TAS features is shown in Figure 5.2.

Using mechanisms from the TSN standards such as the TAS to achieve guaranteed bounded latencies is not as trivial as a simple configuration. Network engineers need to impose restrictions in the network to guarantee a quality of service to the time-critical data flows. These restrictions result in the network queues being in some deterministic (or maximally bound) state that allows reasoning about how specific data flows will be delayed in the network. In this thesis, we will look at a simplified version of the scheduled traffic strategy for achieving bounded latencies which is commonly used for statically computed transmission schedules for time-triggered frames [Craciunas and Oliver 2014]; [Craciunas et al. 2016]; [Dürr and Nayak 2016]. The scheduled traffic strategy involves allocating Ethernet frames from specific flows into dedicated time slots that are configured in the TAS. The TAS will provide isolation be-

tween the time-triggered traffic flows, and all other traffic on the network. To isolate time-triggered flows from each other, the end nodes generating the flows will need to transmit according to a synchronized schedule in their respective time frames. In this thesis, an additional constraint is placed on the strategy such that a flow gets dedicated access to the whole network. This simplifies the assurance case presented, as the focus of this thesis is on an argument structure as opposed the analyses required for more complex cases. The resulting system is one that has sub-optimal bandwidth usage. The inclusion of a more complex strategy is left for future work that builds on the core argument presented in this thesis.

5.2 Building an Assurance Case

Terminology The unidirectional communication from an end node to another through the network is called a *flow*. Here, the realization of a flow will be periodic packets of a static size and format, typical of traditional control system applications. The flows considered here have hard real-time requirements. To meet these requirements, specifications regarding communications timing are determined during system design. Specifically, the timing specification includes upper bounds for the *end-to-end latency* and *jitter*, as well as a *period* for packets contained in this flow.

Top level claim A top-down approach is used to construct this assurance case, and the argumentation will be represented in the core Goal Structuring Notation (GSN) [The Assurance Case Working Group 2021] syntax. The assurance case is shown in Fig. 5.3. The top-level claim refers to a specific flow and its associated timing specification, *FLOW_X* and *TSPEC_X*, respectively. The

top level claim, **G1**, is stated as follows: "All messages in communication flow *FLOW_X* satisfy the timing specification *TSPEC_X*." The associated context (**C1**, **C2**, and **C4**) explains the necessary definitions and presents context regarding the network.

Satisfying the Timing Specification The top level goal can be decomposed into subclaims that individually address latency (**G2**), jitter (**G3**), and period (**G4**).

Arguing Latency Upper Bound We focus on **G2** and leave the jitter and period subclaims as undeveloped goals. Arguing this claim involves analyzing the strategy used for achieving bounded latency in the network. The strategy is based on TSN mechanisms being configured on all devices, and on constraints regarding the timing of packet transmissions. This network achieves bounded latency by scheduling packets from flows to be sent and forwarded through the network at specific time intervals. This ensures that packets from other flows do not interact with each other, by using a TDMA scheme.

Using TAS to Achieve Bounded Latency The TAS as described in the standard only gives the structure and behavior of the mechanism itself. The standard does not specify how the TAS should be used to achieve the bounded latency. It is up to the designer to configure the TAS such that it achieves a bounded latency for specific traffic. Creating a schedule is not a simple task [Stuber et al. 2023] and it requires tool support. In addition to analyzing the constraints on the schedule itself, the AC also exposes the requirements on other participating devices in the network. For example, if the schedule design assumes that a node will transmit a packet at a specific time, this should be captured in the argument, since the argument of bounded latency (below) relies on this behaviour.

Bound Latency Strategy The strategy is to assign each flow a time slot in which no other traffic in the network can be forwarded. This means that during the time slot dedicated to *FLOW_X*, only packets from *FLOW_X* will be transmitted throughout the network. However, each traffic class can contain several flows, and the TAS is limited to isolating only traffic classes, not the flows within the classes. This means that the TAS itself can not prevent other flows from interfering with *FLOW_X*. As such, it is up to the transmitting nodes to ensure that they transmit at the correct time.

Bounded Latency Argument Assuming a system designed according to the strategy defined above, the claim that the latency of *FLOW_X* is bounded within specification can be decomposed into sub-claims. The supporting sub-claims will follow the structure of the constraints in the strategy above, in which the packets of *FLOW_X* are assigned specific time slots within the TAS schedule. In Fig. 5.3, **S1** states this approach. The TSN strategy is then analyzed to extract what implicit claims are stated in the design. This approach makes implicit claims of the design explicit in the assurance case, to allow for more structured argumentation that the design achieves its goal of a bounded latency. For example, this TSN strategy is dependant on all end nodes transmitting precisely when they are allowed to according to the network schedule. This dependency is explicit in the assurance case, and ensures that the reasoning also contains an argument that the end nodes will indeed transmit precisely when allowed.

Decomposing **G2** according to **S1**, the TSN design was found to be dependant on the three subclaims **G5**, **G6**, and **G7**. We next focus on **G6**. **G6** is required so that *FLOW_X* messages sent at the correct time will arrive within the specified latency. **G6** can be further decomposed into a series of support-

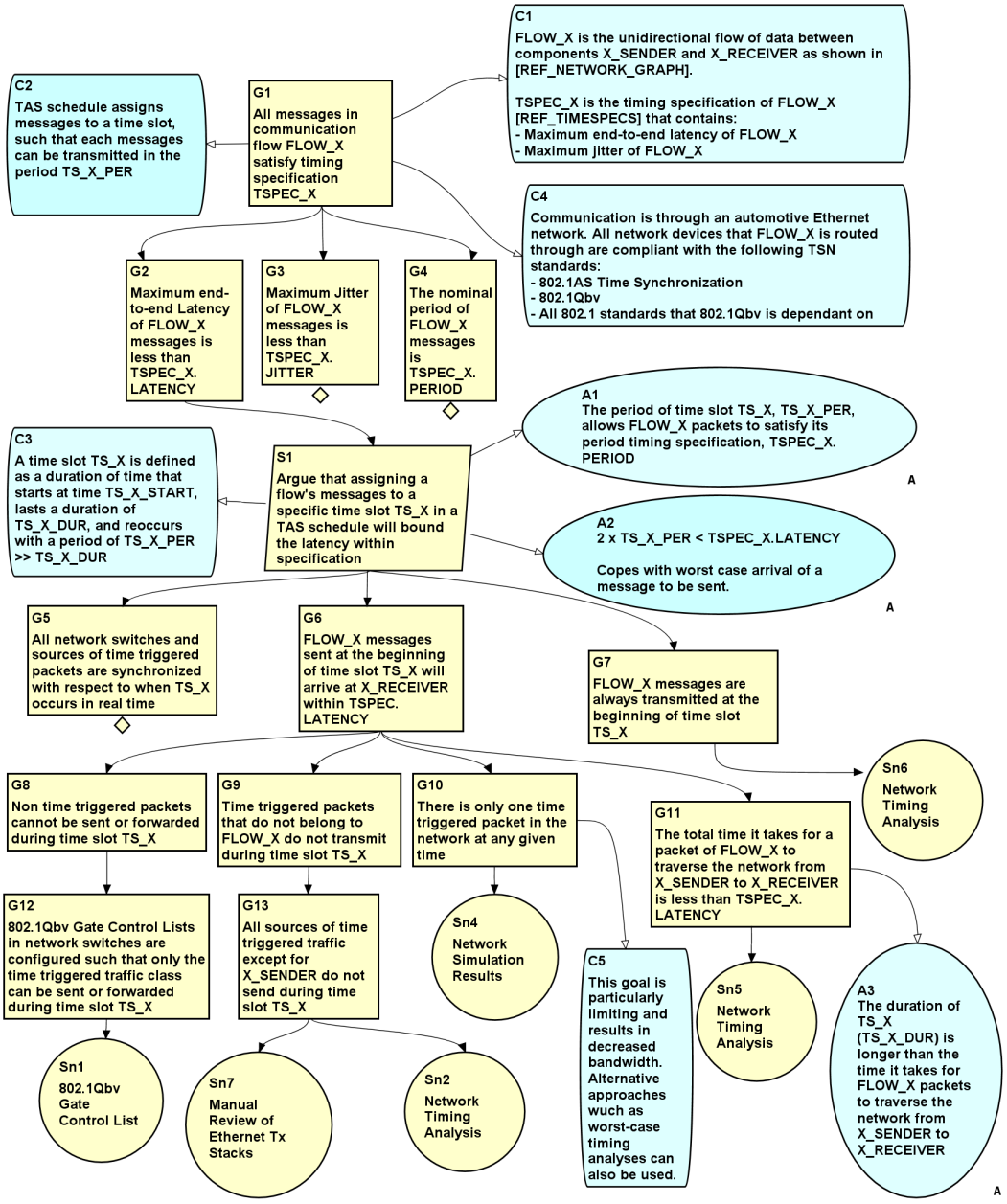

Figure 5.3: Argument for the TSN network's ability to satisfy the timing requirements of a specific flow in the network. This argument in its current state is not complete and serves more as a pattern than a completed argument. Material from: 'Kapinski et al, *Assurance Cases for Timing Properties of Automotive TSN Networks*. Computer Safety, Reliability, and Security. SAFECOMP 2023 Workshops. SAFECOMP 2023. Lecture Notes in Computer Science, vol 14182. Published 2023, Springer, Cham.' [Kapinski et al. 2023]

ing goals **G8**, **G9**, **G10**, and **G11**. **G8** states that time triggered traffic is isolated temporally from non time triggered traffic during *TS_X*. This isolation is directly supported by the gate control list of the TAS, and can be verified to show that during *TS_X* only *FLOW_X* traffic can be forwarded and sent. **G9** is needed for similar reasons as **G8**, but is more complicated as the TAS does not isolate messages within the same traffic class, which in this case is time triggered packets. It is then up to all sources of time triggered packets that they do not send any packets during *TS_X*, which is a process that can be controlled by either SW or HW on every time triggered node. To support **G9**, **G13** states that it is up to the transmitting nodes to ensure that they do not transmit during *TS_X*. For the sake of this argument, **G13** is supported by an empirical timing analysis (**Sn2**) to show that the nodes do not transmit when they should not, and a manual review of the HW/SW stack (**Sn7**) to ensure that the mechanism that does not allow them to transmit at the wrong time accomplishes this task.

5.3 Conclusion

In this chapter a partial assurance case is presented that shows the logical decomposition of goals that a system using TSN mechanisms must satisfy to achieve bounded network latency. The purpose of creating this assurance case is to investigate what evidence is required to show that a particular system using these TSN mechanisms will achieve its real-time networking requirements, specifically, that the latency of certain messages will be less than a specified value. Using GSN, an argument is formed to support the claim that the network latency of a specific message will be bound by a specified maximum value.

This top level claim is decomposed into sub-goals until actual evidence can be referenced, or until a sub-goal is too complex to analyze within the scope of this thesis. The resulting assurance case shows the different types of evidence / sub-goals that are required in order to assure that the message will pass through the network within a bounded latency. It is shown in this case which specific behaviours will need to be achieved by the TSN mechanisms in order to satisfy the guaranteed latency requirement.

One potential use of this work is to inform the further development of patterns and templates for reasoning about timing properties of systems using these TSN mechanisms. In automotive E/E development the functional safety of the resulting system is very important. As part of managing functional safety, a safety case can be used to manage the overall reasoning for why the designers believe that their system is safe. Managing the safety case is an important part of compliance with software safety standards, and aids incremental certification. This work is intended to be part of a vehicle's overall safety case, where safety hazards are identified to be caused by a failure of the network to meet latency requirements. The safety case will need to argue that these hazards are mitigated by using TSN mechanisms, using an argument structure such as the one presented in this chapter.

Chapter 6

Conclusion

In this thesis, networking aspects of migrating to a centralized automotive architecture have been analyzed. Using the constructed prototype as a target platform to modify existing powertrain software, several methods of migrating signal-oriented CAN communications to CAN-FD and Ethernet have been presented. The transition to Ethernet not only includes the addition of significant middleware to enable the full stack of protocols, but the design of the network itself is much more complex due to the real-time capabilities being provided by extensions as opposed to being built in to the lower layers as it is in CAN and CAN-FD. In the case of gPTP, the fundamental service of time synchronization which the Time Aware Shaper is dependant on, is provided by a mixture of hardware mechanisms and a software stack. This added complexity is a key focus in this thesis due to the fact that low-latency and deterministic communication is very important in safety-critical applications such as those found in the powertrain domain.

To help navigate this complexity, this thesis presents a partial assurance case which provides the structure of an argument that the TSN standards

can provide bounded low-latency communications with time-triggered communication. The completion of such an argument for a production system is important if the safe operation of the vehicle is dependant on low-latency communications provided by TSN. The argument provided can be used in safety analyses that involve hazards caused by Ethernet messages being delayed beyond their designed latency. The argument does not provide direct proof that the TSN mechanisms will provide bounded latency. It should be used as a required argument that must be satisfied for a particular system if TSN is to be used to provide real-time capabilities to safety-critical components. By using an assurance case such as the one presented early on in system design, it can help designers create technical requirements that support the argument, or consider a different design if it is determined that the latencies required can not be guaranteed.

The work presented in this thesis can be applied by an assortment of different stakeholders in the development of centralized automotive E/E systems. The analysis of migrating software can be used by OEM system architects looking to re-use legacy ECUs in centralized architectures. This work provides considerations that they should take into account during the migration, as well as some of the options they have. The section on TSN also shows what additional software and hardware configurations need to be taken into account in order to provide real-time communications using Ethernet. The migration analysis can also be used by tier-2 suppliers looking to provide hardware and software support for centralized architectures. Hardware manufacturers will need to include TSN support, and software providers should include software for managing, configuring, and implementing the TSN mechanisms. The results of this thesis also shows the need for tool support for architectural design

relating to Ethernet in order to optimize network usage, and evaluate what types of migrations are possible given the legacy ECUs being migrated, and the desired platform architecture.

The partial assurance case is expected to be used by OEM functional safety engineers and network engineers as a working document that contains the core argument that real-time communications are provided. The assurance case can be iterated upon to include system specific details, and can be adapted to the network engineer's specific strategy while still using the same core concepts presented in this thesis. For functional safety assurance, this case can be used to find fault conditions for hazards related to communication failures. The assurance case shows exactly which goals must be satisfied to provide latency guarantees, and therefore any fault that affects the validity of the goals in this case need to be mitigated for the safety case to be valid. This partial assurance case is meant to be a starting point for functional safety engineers and network engineers to iterate upon, adding the specific details of their systems. Future work on this topic is to provide examples of cases with more complex strategies that include formal timing analyses as the foundation for bounded latency. The expected differences depend on the strategy, but would involve showing that the model used in the analysis is accurate and that all assumptions made in the analysis are true. Similar to the assurance case presented in this thesis, these assumptions can be shown to be true using TSN mechanisms that restrict the traffic such that certain assumptions can be reasonably made.

Bibliography

802.1, IEEE (n.d.). *Time Sensitive Networking (TSN) Task Group*. URL: https://1.ieee802.org/tsn/ (visited on 02/21/2024) (cit. on p. 13).

Apostu, Silviu et al. (Apr. 2019). *Automotive software andelectrical/electronicarchitecture: Implications for OEMs*. Tech. rep. McKinsey Center for Future Mobility (cit. on p. 1).

AUTOSAR Consortium (2020). *SOME/IP Protocol Specification, Release R20-11*. https://www.autosar.org/fileadmin/user_upload/standards/foundation/20-11/AUTOSAR_PRS_SOMEIPProtocol.pdf. [Online; accessed October 12th, 2021] (cit. on p. 35).

Bandur, V. et al. (2021). "Making the Case for Centralized Automotive E/E Architectures". In: *IEEE Transactions on Vehicular Technology* 70.2, pp. 1230–1245 (cit. on pp. 2, 34).

Bandur, Victor et al. (June 2021). "A Safety Architecture for Centralized E/E Architectures". In: *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE (cit. on p. 50).

Bandur, Victor et al. (Mar. 2022). "Aspects of Migrating from Decentralized to Centralized E/E Architectures". In: *SAE Technical Paper Series*. AN-

NUAL. SAE International. URL: http://dx.doi.org/10.4271/2022-01-0747 (cit. on pp. 25, 27, 28, 30, 32).

Berisa, Aldin et al. (May 2023). "Investigating and analyzing CAN-to-TSN gateway forwarding techniques". In: *2023 IEEE 26th International Symposium on Real-Time Distributed Computing (ISORC)*. Nashville, TN, USA: IEEE (cit. on p. 22).

Bordoloi, Unmesh D. and Soheil Samii (2014). "The Frame Packing Problem for CAN-FD". In: *2014 IEEE Real-Time Systems Symposium*, pp. 284–293 (cit. on p. 33).

Craciunas, Silviu S. and Ramon Serna Oliver (2014). "SMT-based Task- and Network-level Static Schedule Generation for Time-Triggered Networked Systems". In: *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*. RTNS '14. Versaille, France: Association for Computing Machinery, 45–54. ISBN: 9781450327275. URL: https://doi-org.libaccess.lib.mcmaster.ca/10.1145/2659787.2659812 (cit. on p. 52).

Craciunas, Silviu S. et al. (2016). "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks". In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. ACM (cit. on pp. 23, 52).

Dürr, Frank and Naresh Ganesh Nayak (2016). "No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN)". In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. RTNS '16. Brest, France: Association for Computing Machinery, 203–212. ISBN: 9781450347877. URL: https://doi-org.libaccess.lib.mcmaster.ca/10.1145/2997465.2997494 (cit. on p. 52).

Farkas, Janos (2018). *IEEE 802.1 TSN TG Overview*. https://www.ieee802.org/1/files/public/docs2018/detnet-tsn-farkas-tsn-overview-1118-v01.pdf. Accessed: 2024–02-24 (cit. on p. 13).

Geyer, Fabien and Georg Carle (2016). "Network engineering for real-time networks: comparison of automotive and aeronautic industries approaches". In: *IEEE Communications Magazine* 54.2, 106–112 (cit. on p. 49).

Gopu, G. L., K. V. Kavitha, and James Joy (2016). "Service Oriented Architecture based connectivity of automotive ECUs". In: *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pp. 1–4 (cit. on p. 2).

Graydon, Patrick and Iain Bate (2014). "Realistic Safety Cases for the Timing of Systems". In: *The Computer Journal* 57.5, 759–774 (cit. on p. 49).

Hank, Peter et al. (2013). "Automotive Ethernet: In-vehicle networking and smart mobility". In: *2013 Design, Automation and Test in Europe Conference and Exhibition*, pp. 1735–1739 (cit. on p. 1).

IEEE (2008). *1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. [Online; accessed September 8th, 2020]. URL: https://standards.ieee.org/standard/1588-2008.html (cit. on p. 43).

— (2015). *IEEE 802.1Qbv-2015 – IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment 25: Enhancements for Scheduled Traffic*. Tech. rep. IEEE 802.1Qbv. IEEE (cit. on pp. 15, 44, 51).

— (2016a). "IEEE Standard for a Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks". In: *IEEE Std 1722-2016 (Revision of IEEE Std 1722-2011)*, pp. 1–233 (cit. on p. 38).

— (2016b). "IEEE Standard for Ethernet Amendment 1: Physical Layer Specifications and Management Parameters for 100 Mb/s Operation over a Single Balanced Twisted Pair Cable (100BASE-T1)". In: *IEEE Std 802.3bw-2015 (Amendment to IEEE Std 802.3-2015*, 1–88 (cit. on p. 12).

IEEE (2017a). *802.1CB – Frame Replication and Elimination for Reliability*. [Online; accessed October 29th, 2019]. URL: https://1.ieee802.org/tsn/802-1cb/ (cit. on p. 45).

— (2017b). *802.1Qci – Per-Stream Filtering and Policing*. [Online; accessed October 29th, 2019]. URL: https://1.ieee802.org/tsn/802-1qci/ (cit. on p. 45).

— (2017c). *Time-Sensitive Networking Task Group*. [Online; accessed October 29th, 2019]. URL: http://www.ieee802.org/1/pages/tsn.html (cit. on p. 43).

— (2018). *802.1Q-2018 - IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks*. [Online; accessed September 8th, 2020]. URL: https://standards.ieee.org/standard/802_1Q-2018.html (cit. on p. 46).

— (2019). *802.1AS-Rev – Timing and Synchronization for Time-Sensitive Applications*. [Online; accessed October 29th, 2019]. URL: https://1.ieee802.org/tsn/802-1as-rev/ (cit. on pp. 14, 43).

— (2020). "IEEE Standard for Local and Metropolitan Area Networks-Timing and Synchronization for Time-Sensitive Applications". In: *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, 1–421 (cit. on p. 14).

— (2022). "IEEE Standard for Ethernet". In: *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, 1–7025 (cit. on p. 36).

— (2024). "IEEE Draft Standard for Time-Sensitive Networking Profile for Automotive In-Vehicle Ethernet Communications". In: *IEEE P802.1DG/D4.0, July 2024*, pp. 1–61 (cit. on p. 14).

ISO (2011). *ISO 26262: Road Vehicles – Functional Safety* (cit. on p. 8).

Jang, Hyuksoo et al. (2023). "Design of a Hybrid In-Vehicle Network Architecture Combining Zonal and Domain Architectures for Future Vehicles". In: *2023 IEEE 6th International Conference on Knowledge Innovation and Invention (ICKII)*, pp. 33–37 (cit. on p. 1).

Kapinski, Ryan et al. (2023). "Assurance Cases for Timing Properties of Automotive TSN Networks". In: *Computer Safety, Reliability, and Security. SAFECOMP 2023 Workshops: ASSURE, DECSoS, SASSUR, SENSEI, SRToITS, and WAISE, Toulouse, France, September 19, 2023, Proceedings*. Toulouse, France: Springer-Verlag, 26–31. ISBN: 978-3-031-40952-3. URL: https://doi.org/10.1007/978-3-031-40953-0_3 (cit. on pp. 50–52, 56).

Kern, A, T Streichert, and Jurgen Teich (2011). "An automated data structure migration concept - From CAN to Ethernet/IP in automotive embedded systems (CANoverIP)". In: *2011 Design, Automation & Test in Europe*. IEEE (cit. on p. 39).

Kern, Andreas et al. (2011). "Gateway Strategies for Embedding of Automotive CAN-Frames into Ethernet-Packets and Vice Versa". In: *Architecture of Computing Systems - ARCS 2011*. Ed. by Mladen Berekovic et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 259–270. ISBN: 978-3-642-19137-4 (cit. on p. 21).

Lee, Young Seo et al. (2016). "Automotive ECU Software Reprogramming Method Based on Ethernet Backbone Network to Save Time". In: *Pro-*

ceedings of the 10th International Conference on Ubiquitous Information Management and Communication. IMCOM '16. Danang, Viet Nam: Association for Computing Machinery. ISBN: 9781450341424. URL: https://doi-org.libaccess.lib.mcmaster.ca/10.1145/2857546.2857586 (cit. on p. 1).

Maile, Lisa, Kai-Steffen Hielscher, and Reinhard German (2020). "Network Calculus Results for TSN: An Introduction". In: *2020 Information Communication Technologies Conference (ICTC)*, pp. 131–140 (cit. on p. 24).

Maul, Mario, Gerhard Becker, and Ulrich Bernhard (Feb. 2018). "Service-oriented EE zone architecture key elements for new market segments". In: *ATZelektronik worldwide* 13.1, pp. 36–41. ISSN: 2192-9092 (cit. on p. 1).

Mauser, Lucas, Stefan Wagner, and Peter Ziegler (2023). "Methodical Approach for Centralization Evaluation of Modern Automotive E/E Architectures". In: *Software Architecture. ECSA 2022 Tracks and Workshops: Prague, Czech Republic, September 19–23, 2022, Revised Selected Papers*. Prague, Czech Republic: Springer-Verlag, 165–179. ISBN: 978-3-031-36888-2. URL: https://doi.org/10.1007/978-3-031-36889-9_13 (cit. on p. 1).

Sandstrom, K., C. Norstom, and M. Ahlmark (2000). "Frame packing in real-time communication". In: *Proceedings Seventh International Conference on Real-Time Computing Systems and Applications*, pp. 399–403 (cit. on p. 33).

Sinha, Amit Kumar and Shubham Saurabh (2017). "CAN FD: performance reality". In: *2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT)*. IEEE, pp. 1–6 (cit. on p. 30).

Staron, Miroslaw (2021). "Contemporary Software Architectures: Federated and Centralized". In: *Automotive Software Architectures: An Introduction*. Cham: Springer International Publishing, pp. 55–66. ISBN: 978-3-030-65939-4. URL: https://doi.org/10.1007/978-3-030-65939-4_3 (cit. on p. 1).

Stoll, Hannes et al. (2021). "Dynamic Reconfiguration of Automotive Architectures Using a Novel Plug-and-Play Approach". In: *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, pp. 70–75 (cit. on p. 1).

Stuber, Thomas et al. (2023). *A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN)*. arXiv: 2211.10954 [cs.NI] (cit. on p. 54).

Stähle, Hauke et al. (2013). "Towards the deployment of a centralized ICT architecture in the automotive domain". In: *2013 2nd Mediterranean Conference on Embedded Computing (MECO)*, pp. 66–69 (cit. on p. 1).

The Assurance Case Working Group (2021). *Goal Structuring Notation Community Standard Version 3* (cit. on pp. 17, 53).

Vetter, Andreas et al. (2020). "Development Processes in Automotive Service-oriented Architectures". In: *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, pp. 1–7 (cit. on p. 1).

Zhou, Andy et al. (2021). *Software-Defined Vehicles - A Forthcoming Industrial Evolution*. Tech. rep. Deloitte (cit. on p. 1).

Zhu, Hailong et al. (2021). "Requirements-Driven Automotive Electrical/Electronic Architecture: A Survey and Prospective Trends". In: *IEEE Access* 9, pp. 100096–100112 (cit. on p. 2).

Zinner, Helge et al. (June 2011). "Application and realization of gateways between conventional automotive and IP/ethernet-based networks". In: *Pro-*

ceedings of the 48th Design Automation Conference. San Diego California: ACM (cit. on p. 22).