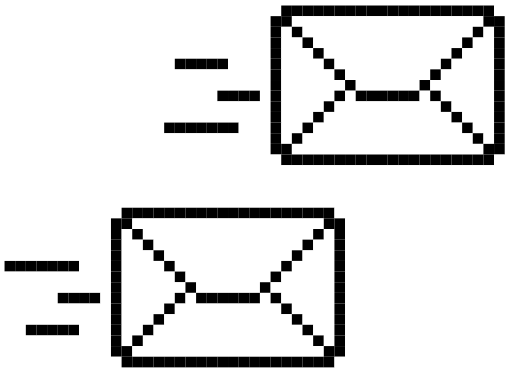
Daniel Thiele

Formal Timing Analysis of Ethernet TSN Extensions for Automotive Applications



Dissertation
Technische Universität Braunschweig

Formal Timing Analysis of Ethernet TSN Extensions for Automotive Applications

Von der Fakultät für Elektrotechnik, Informationstechnik, Physik
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines Doktors

der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von: Dipl.-Ing. Daniel Thiele
aus: Goslar
eingereicht am: 26.11.2021
mündliche Prüfung am: 01.04.2022
1. Referent: Prof. Dr.-Ing Rolf Ernst
2. Referentin: Prof. Ahlem Mifdaoui
Vorsitzender: Prof. Dr.-Ing. Harald Michalik

Druckjahr: 2022

Dissertation an der Technischen Universität Braunschweig,
Fakultät für Elektrotechnik, Informationstechnik, Physik

Kurzfassung

Unfallprävention ist eine der treibenden Kräfte auf dem Weg zum autonomen Fahren. Eine Schlüsselkomponente dazu ist die lückenlose Umfeldwahrnehmung durch eine Vielzahl von Sensoren wie Kameras, Radar, Lidar und Ultraschall. Diese Sensoren erhöhen jedoch die Kommunikationsanforderungen, die an ein Fahrzeugnetzwerk gestellt werden, signifikant und erfordern skalierbare Vernetzungsarchitekturen.

Aufgrund seiner Flexibilität, Geschwindigkeit und Kostenvorteilen wurde Ethernet als Kommunikationsrückgrat moderner Fahrzeugvernetzungsarchitekturen ausgewählt. Eine entscheidende Rolle haben dabei die ausgeklügelten Quality of Service Mechanismen gespielt, die mit den Ethernet AVB und TSN Standards eingeführt wurden und die Integration von unterschiedlich kritischer Kommunkation in einem gemeinsamgenutzten Netzwerk erlauben. Quality of Service Mechanismen stellen allerdings nur den ersten Schritt einer solchen gemischten Integration dar. Nicht minderwichtig ist der Beweis, dass alle Sicherheitsanforderungen, die an die Kommunikation gestellt werden, auch unter widrigen Bedingungen eingehalten werden.

Diese Arbeit fokussiert sich auf letztgenanntes und präsentiert formale Methoden um Performanzgarantien für die Quality of Service Mechanismen von Ethernet TSN abzuleiten. Der Fokus liegt dabei auf IEEE 802.1Qbv (zeitgesteuertes Senden), IEEE 802.1Qch (zyklisches Senden) und IEEE 802.3br (unterbrechbares Senden). IEEE 802.1Qbv reserviert periodisch wiederkehrende Zeitfenster zur Übertragung von kritischen Nachrichten. Es können sich mehrere Nachrichten wenige Fenster teilen oder einzelne Nachrichten individuellen Fenstern zugeordnet werden. Eine Analyse muss die Wechselwirkung von Nachrichten innerhalb eines Zeitfensters als auch den Einfluss der Fenster auf unkritische Nachrichten abdecken. IEEE 802.1Qch hat eine vereinfachung der Latenzberechnung durch zyklisches Senden als Ziel. Durch eine Analyse muss sichergestellt werden, dass die einzelnen Zyklen ohne Überlast eingehalten werden. Mit IEEE 802.3br wird die Zeit, die kritische Nachrichten aufgrund von einer sich in Übertragung befindlichen Nachricht warten müssen, reduziert. Eine Analyse muss sowohl nachweisen, dass sich die Latenzen von kritischen Nachrichten verbessern als auch die negative Auswirkung auf wenigerkritische Nachrichten ausweisen. Die Analysen gliedern sich nahtlos in das Compsitional Performance Analysis Framework ein, das dazu entsprechend erweitert wurde.

Abstract

Safety is one of the main drivers for the transition to autonomously driving vehicles. Thorough environment perception is a key component of such vehicles and requires a multitude of sensors, such as cameras, radars, lidars, and ultrasonic sensors, along with potent computation resources. These sensors significantly increase the communication demand of on-board networks and require scalable automotive communication architectures.

Ethernet has been chosen as the main communication backbone for modern automotive architectures due to its flexibility, high-speed, and cost-effectiveness. A key aspect of Ethernet are its advanced quality of service mechanisms that have been introduced with the Ethernet AVB and TSN standards and allow the integration of various communication nodes with different communication and safety requirements on top of a shared backbone network. However, quality of service mechanisms are only the first step to enable the integration of communication with different safety requirements. The second, equally important, step is the proof that, in a concrete network, safety requirements are met even under worst-case conditions.

This thesis focuses on the latter and presents formal methods to compute performance guarantees for Ethernet TSN's quality of service mechanisms. In particular, formal analyses for IEEE 802.1Qbv (enhancements for scheduled traffic), IEEE 802.1Qch (cyclic queueing and forwarding), and IEEE 802.3br (frame preemption) are presented. IEEE 802.1Qbv reserves periodically-repeating slots in which latency-critical traffic is transmitted. With this scheme, multiple Ethernet frames can share a few of these slots or single frames can be mapped to individual slots. The formal analysis computes both the contention inside these slots as well as the impact of individual slots on other traffic. IEEE 802.1Qch aims to simplify latency computation by forwarding frames in a peristaltic manner. Here an analysis is required to proof the absence of overload, which would impair cyclic forwarding. Finally, IEEE 802.3br reduces the time critical traffic has to wait on the ongoing transmission of other traffic until it can be sent. A formal analysis is required to proof the improvement on critical traffic, while also outlining the degradation of less-critical traffic, due to preemption overhead. The analyses presented in this thesis are based on a compositional performance analysis framework, which has been extended accordingly.

Contents

Chapter 1

Introduction

Fully autonomous driving is the ambitious goal of the automotive industry. There are a number of reasons for this. The main one is safety. Over the past 20 years, vehicle safety features and assistance systems helped to cut down the number of traffic deaths significantly despite a growing number of motor vehicles, e.g. [160]. However, statistics show that, in the European Union for instance, 90 % of accidents (fatal and injuries) are caused by human error [35]. Consequently, it is desirable to (ultimately) replace the human driver. In order to do so, automotive systems must transition from advanced driver assistance systems (ADAS) towards autonomously-driving vehicles [59], [152]. This transition is being pushed by governments, such as the European Commission, mandating more and more assistance system for new vehicles [35]. The European Union's long-term goal is to even have nearly zero traffic deaths by 2050 [35]. In addition to increased safety, autonomous driving is also expected to lead to more traffic efficiency, enabling smooth traffic flow, less emissions, and higher lane capacity.

One of the key requirements to realize fully autonomous driving is that vehicles have a 360 degree perception of their environment. This is realized via an increasing number of different sensors, including multiple camera, radar, lidar, and ultrasonic sensors and the fusion of their data. It is even possible to extend the environmental perception beyond a vehicle's own sensoric horizon by exchanging information with other vehicles and infrastructure (roadside as well as backend data centers).

All these sensors and communication use cases put new requirements on the E/E architecture. Environment perception sensors are typically distributed across the vehicle and often stream their information at high data rates. In order for an autonomous system to react in a timely manner to its environment, information must be communicated in real-time. With the higher levels of autonomous driving, redundancy is required to implement fail-operational modes [152]. In addition to increased communication from environmental sensors, Car-to-X communication will also introduce new traffic patterns to efficiently exchange information with other vehicles and infrastructure (including bulk traffic such as software updates over the air). An important challenge is to open

up the in-vehicle communication network to also support customer (Internet) traffic, resulting in mixed-critical communication requirements. Ethernet has been chosen as the communication backbone technology to accommodate all these communication use cases.

In the IT domain, Ethernet is the dominating communication technology, with support for basic quality of service. Since the Ethernet AVB standardization advances, Ethernet has been increasingly considered for non-IT use cases. With the advent of automotive-grade Ethernet PHYs (see next section) the last hurdle for using Ethernet as the communication backbone of automotive E/E architectures has been overcome. Ethernet enables high bandwidth, packet switched communication. In switched Ethernet networks, the collision domains are reduced to the link between devices, which allows a significantly better utilization of the communication medium (different parts of the network can be used concurrently) compared to bus-based (shared medium) communication technologies such as CAN or FlexRay. As a network, Ethernet can scale easily by adding further switches and increasing link speeds. Over the past years, Ethernet introduced elaborate quality of service mechanisms, addressing different use cases (latency, redundancy, bandwidth, mixed-critical traffic, etc.), allowing real-time communication. This also includes support for monitoring/policing individual traffic streams, which allows more robust networks as well as better network utilization by only enforcing quality of service policies when necessary (cf. Section 1.3). The support of different speed grades, ranging from 10 MBit/s to multiple GBit/s, offers the potential to unify the entire in-vehicle communication infrastructure in the long run. As Ethernet is used in multiple domains with similar requirements, e.g. industrial automation and avionics, standardization and development effort is shared among all stakeholders, making Ethernet a comparatively cost-effective solution. Furthermore, Ethernet and its related higher-layer communication protocols (such as IP and TCP or UDP) allow seamless integration with existing tools and IT infrastructure, including the availability of trained personnel.

1.1 Evolution of Automotive E/E Architectures[1]

E/E architectures have a huge impact on multiple areas of vehicle design such as safety, security, and cost. As E/E architectures are typically designed a few years ahead of series production, the challenge they face is the consideration of the current state of the art as well as the future availability and evolution of components, while also taking into account paradigm shifts in development, computation, and communication.

An actual challenge is the development of a new E/E architecture supporting the increasing number of in-vehicle sensors and actuators as well as providing the computational power to cover demanding ADAS and infotainment use cases. Figure 1.1 shows the conceptual evolution of automotive E/E architectures. Note that there is no common E/E architecture, as each OEM implements its

[1]This section provides a summary from public presentations and discussions the author of this thesis was engaged in, as well as literature, focussing on publications from OEMs and tier-1 suppliers: [178], [186], [59].

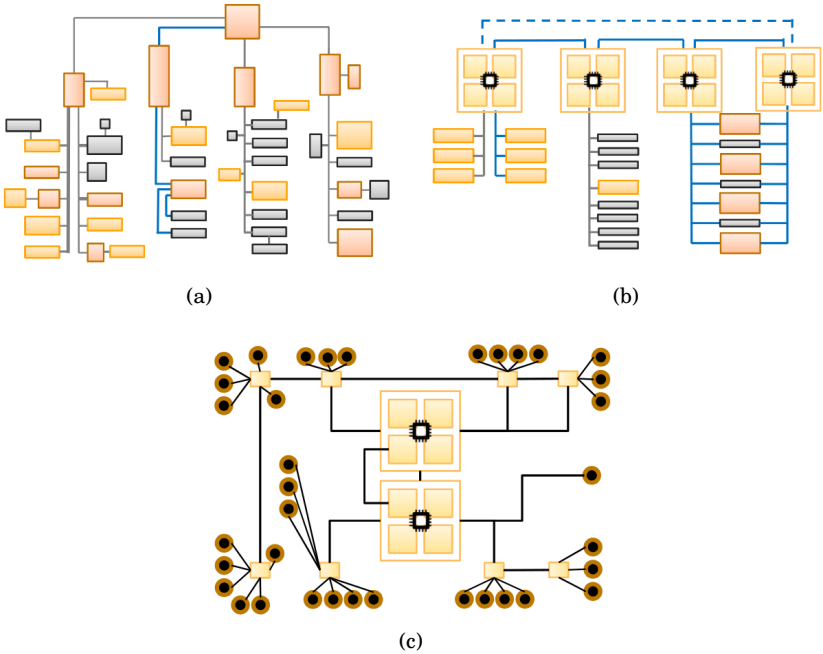


Figure 1.1: Evolution of conceptual automotive E/E architectures: (a) traditional architecture, (b) domain architecture, and (c) centralized architecture (source: [186]).

own characteristic architecture. There are, however, conceptual similarities, which are highlighted by the architectures in the figure.

1.1.1 Traditional Architecture

Traditional E/E architectures, as shown on Figure 1.1a, are function-oriented communication architectures. They comprise application-specific ECUs and buses. The ECUs are mainly small and efficient microcontrollers. The communication infrastructure is purposely-built, typically including several LIN, CAN, or FlexRay buses. Computation as well as communication is very static and predictable and mainly signal-based. If Ethernet is used at all, its deployment is very limited. The strength of this architecture is its predictability, which makes it suitable for time-critical real-time applications. At the same time, this is also its weakness, as adaptation and evolution require significant effort.

1.1.2 Domain Architecture

The domain E/E architecture is shown in Figure 1.1b. Here, the architecture is partitioned into different domains. Each domain has a performant domain controller ECU and the domain controllers are interconnected via a performant interconnect, such as Ethernet. The domain controllers allow functional consolidation and contain intra-domain communication within the respective domain. The domain architecture is anticipated to have Ethernet communication with 10 to 50 links and speed requirements between 100 MBit/s and 10 GBit/s. The domain architecture is often considered as an intermediate step towards the centralized architecture.

1.1.3 Centralized Architecture

In Figure 1.1c, the centralized architecture, as the next step in the evolution of automotive E/E architectures, is shown. It marks a tuning point in automotive E/E architectures, as it is designed to allow the introduction of many IT and consumer electronics technologies, in particular high performance processors and high performance networks. The computation is further consolidated to a (few) powerful central computing node(s), often executing virtualized ECUs. These powerful nodes are required to cover the increasing computational demands of, for instance, ADAS (e.g. sensor fusion) and infotainment applications. In order to achieve their performance, these computation nodes implement dynamic microarchitectures (including caches, speculative execution, etc.) that are (from a timing perspective) less predictable than microcontrollers with traditional embedded microarchitectures. Likewise, these processors will not run traditional automotive operating systems, but rather dynamic operating systems like Linux or QNX. Another feature taken from consumer electronics is support for remote software updates.

Ethernet will be used extensively as a communication backbone with over 50 links and data rate requirements of up to 50 GBit/s. Redundancy for fail-operational support is introduced where needed. The high link count and data rate demand is due to the increasing number of IP-capable sensors and actuators that these E/E architectures need to support as well as the increased connectivity (e.g. Car-to-X or Cloud connectivity). Service-oriented communication will be used to handle the complexity of the (naturally) distributed system of in-vehicle sensors and actuators.

There are different implementation variants of the centralized E/E architecture, with the ultimate goal to become a zonal architecture, which introduces another level of hierarchy below the central computing nodes: zonal ECUs. Zonal ECUs are mediators between the central controller on their northbound interface and sensors and actuators on their southbound interface. Network communication will become hierarchical with zonal ECUs aggregating low speed data from (standardized) sensors and forwarding it to the central computing node and vice versa for the communication from the central communication node to actuators.

All automotive E/E architectures have in common that they need to support quality of service for the increasing variety of communication use cases.

1.2 Quality of Service in Ethernet

Quality of service (QoS), i.e. the ability to treat traffic streams according to their performance requirements, has always been an important property of traditional in-vehicle buses such as CAN [149] and FlexRay [94]. This applies even more to Ethernet, as the upcoming centralized and zonal automotive E/E architectures with their increasing number of interconnected ECUs and service-oriented architectures will introduce more communication. With these architectures, we also see an increased diversity of traffic classes, ranging from latency-sensitive control loops over bandwidth-demanding driver assistance traffic to best effort consumer traffic.

Initially, Ethernet implemented shared medium communication with a CSMA/CD mechanism for medium access. There was no QoS and the timing was undeterministic due to CSMA/CD's exponential backoff mechanism to recover from collisions on the medium. Since then, however, Ethernet has evolved into a switched network, where the points of contention are located in dedicated active network components, i.e. switches. There, packets can be queued until a transmission selection mechanism resolves the contention.

Over the years, a number of QoS mechanisms have been implemented in Ethernet. IEEE 802.1Q standardized non-preemptive static priority based transmission selection in Ethernet [76]. Weighted round robin, although not standardized by the IEEE, is an alternative transmission selection mechanism that is often available in switches.

When the audio/video community was looking for a way to consolidate their networks (supporting time critical audio/video traffic as well as best effort traffic) and reduce wiring cost, Ethernet was chosen as the communication medium and Ethernet AVB has been standardized by the IEEE. Ethernet AVB mainly addresses the requirements of professional audio/video equipment and introduces time synchronization [69], QoS in the form of traffic shaping for bandwidth control [72], and resource reservation [71].

Ethernet TSN widens the application domain of IEEE-standardized Ethernet even further into the industrial automation and automotive domains by enhancing time synchronization [77] and introducing more QoS mechanisms (including scheduled, i.e. time-triggered, traffic transmission) [83], [80], [82], frame preemption [3], [79], per-stream filtering and policing [81], redundancy [78], as well as the correspondingly enhanced resource reservation mechanisms [86], [87].

Outside of IEEE standardization, lower-volume application domains defined their own set of Ethernet standards, often with custom extensions and features to meet domain-specific requirements. In particular, there are Profinet in the industrial automation domain [36], ARINC 664 Part 7 (AFDX) in the avionics domain [6], and TTEthernet [14].

Regardless of which QoS mechanisms are used in a network, the higher the criticality of a traffic stream, the more predictable its performance (e.g. latency) must be.

1.3 Performance Evaluation of Ethernet Networks

There are different ways to evaluate the performance of an Ethernet network. The two main approaches are simulation and formal performance analysis.

Simulation applies a stimulus (or multiple stimuli) for a given amount of time to a network. During the simulation the behavior of the network can be observed. Due to this observability, valuable information can be extracted, such as minimum, maximum, and average observed (communication) latency and (switch) resource usage numbers as well as, for instance, histograms of these metrics. The main shortcoming of simulation is to get results with an acceptable confidence level. To reach this confidence, often long simulation runs are required and the stimuli have to be chosen such that they also trigger performance corner cases. This can be quite hard and even be next to impossible especially for (larger) multi-hop networks, as the amount of simulation time and the number of stimuli is limited. There might even be counter-intuitive cases where not sending data in simulation leads to worse scenarios than if this data had been sent (e.g. when a traffic stream, due to packet loss, causes less interference on another traffic stream, which then, interferes more with a third traffic stream) [48]. Consequently, simulation results have to be taken with reservation and can only provide a (still important) lower bound on network performance, often called the observed worst-case behavior.

Formal methods, in contrast to simulation, provide performance guarantees, i.e. upper bounds on the worst-case behavior. For actual networks and systems, however, it is typically computationally prohibitive to analyze their entire state space to derive tight or even exact guarantees. What formal methods typically do instead to manage computational tractability is to construct pessimistic (potentially unrealistic) scenarios that are guaranteed to include the worst-case behavior. The actual degree of the pessimism is often unknown, however.

Consequently, simulation and formal methods are both important during network performance evaluation and complement each other [48], [132]. Formal methods can be used to verify the performance of (time-)critical traffic streams in a network, whose failure to meet certain performance requirements is expensive or leads to severe consequences. Simulation, complementarily, can be used to evaluate the typical performance of the remaining less or non critical traffic streams, where occasional frame drops can be tolerated or are handled by a retransmission mechanism (which typically has a (negative) impact on the timing behavior) such as TCP. In order to support a design process where decisions are selectively based on both formal methods and simulation, the underlying network components (switches, software stacks, etc.) must provide means to provide sufficient independence between critical and non-critical traffic streams. In particular, Ethernet TSN's IEEE 802.1Qbv (enhancements for scheduled traffic) and IEEE 802.1Qci (per-stream filtering and policing) standards aim to support such design processes.

1.4 Research Objective and Contribution

In this chapter, we outlined how Ethernet is a key communication technology to address the communication requirements of current and future automotive E/E architectures. This is mainly due to the advanced QoS mechanisms, that have been introduced in recent Ethernet standards, such as Ethernet TSN, allowing latency control, redundancy, mixed-critical traffic streams, etc. We have identified that both simulation and formal analyses are required to evaluate the performance and dimension in-vehicle E/E architectures early in the design process.

In this thesis, we focus on the latter and present formal performance analyses for IEEE 802.1Qbv (enhancements for scheduled traffic), IEEE 802.1Qch (cyclic queueing and forwarding), and IEEE 802.3br/IEEE 802.1Qbu (frame preemption). We will also compare the performance guarantees from these analyses to the guarantees from priority-based transmission selection according to "standard Ethernet", i.e. IEEE 802.1Q. Additionally, we will provide a workload optimization, supporting multi-speed networks, which provides significantly tighter performance guarantees, especially for multi-hop topologies.

Our formal performance analysis is based on the compositional performance analysis methodology (CPA) [66] and integrates seamlessly with other CPA based analysis approaches, such as CAN and FlexRay or processor scheduling, so that entire cause effect chains can be analysed.

1.5 Organization

This thesis is organized as follows. In Chapter 2, we introduce Ethernet from an automotive perspective. We present Ethernet's QoS mechanisms, typical automotive switch architectures, and higher-layer protocols. We also give a short overview of the various Ethernet AVB and Ethernet TSN standards to complete the picture.

Chapter 3 summarizes the formal compositional performance analysis (CPA) method, on which the Ethernet analyses presented in this thesis are built on. We present CPA's system model as well as its iterative analysis approach. The emphasis is put on the parts required to apply CPA to Ethernet.

Chapter 4 focuses on using CPA to formally analyze automotive Ethernet transmission selection mechanisms as defined by the Ethernet TSN set of standards. In particular, we will focus on the analysis of IEEE 802.1Q, IEEE 802.1Qbv (enhancements for scheduled traffic), IEEE 802.1Qch (cyclic queueing and forwarding), and IEEE 802.3br/IEEE 802.1Qbu (frame preemption). We start by reviewing related work and by showing how the system model of Ethernet can be mapped to CPA's system model. Next, we present the compositional performance analysis of IEEE 802.1Q as a baseline on which we will base our analysis of Ethernet TSN transmission selection mechanisms. We finish this chapter by generalizing how workload correlations on shared output ports can be exploited in our CPA-based approach, which, as we will see in the evaluation, can lead to a significant improvement of the formal timing guarantees.

In Chapter 5 we evaluate our proposed analyses using four different topologies. We show the benefit of exploiting workload correlations and compare IEEE 802.1Q with Ethernet TSN's IEEE 802.1Qbv and IEEE 802.1Qch. Furthermore we evaluate the benefit of frame preemption on IEEE 802.1Q and IEEE 802.1Qbv for two integration modes.

Finally, in Chapter 6, we summarize and conclude this thesis.

Chapter 2

Automotive Ethernet

In order to analyze communication networks from a timing perspective in the system-level context, it is key to understand the entire communication path and its individual components from the communication medium to the application and identify the implication of each involved component on communication timing. In this chapter we provide the technical background of Ethernet-based communication networks in the automotive context.

We start with classification of Ethernet and its related protocols in the ISO/OSI model of layered communication. Then we provide a brief introduction to the physical layer standards, that are used in the automotive context. We discuss Ethernet's frame structure and addressing scheme and put a special emphasis on QoS in Ethernet, where we present the relevant standardization efforts. The chapter is finished by the introduction of a generic Ethernet switch architecture model and a discussion of the higher-layer protocols that are used on top of Ethernet.

2.1 Introduction to Ethernet

Ethernet is standardized as part of the IEEE 802 project. This project defines a set of standards addressing local and metropolitan area networks on the physical and data link layers of the ISO/OSI model [92]. IEEE 802 is split into several work groups, each of which focuses on a certain networking technology (e.g. Ethernet) or a certain networking aspect (e.g. quality of service).

This section serves as a formal introduction to Ethernet. We present how Ethernet fits into the ISO/OSI communication model and its addressing scheme along with its frame format.

2.1.1 Ethernet in the ISO/OSI Model

The open systems interconnection (OSI) reference model (ISO/OSI model) [92] subdivides network-based communication into a layered model. It is a theoretical framework that breaks down network communication into different

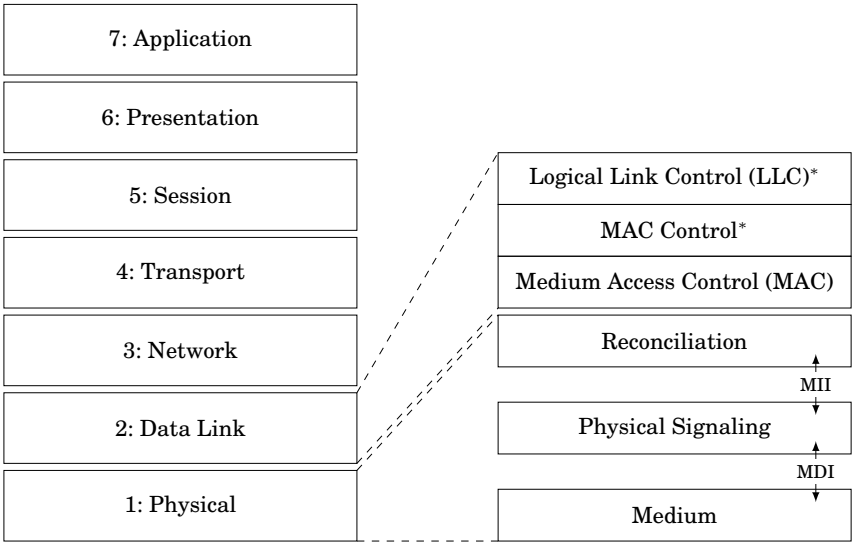


Figure 2.1: Ethernet in the ISO/OSI reference model. Layers marked with * are optional.

layers without defining their actual implementation. Each layer has a distinct responsibility regarding communication and provides certain functionality to its higher layers, while building on the functionality of its lower layers. This modularity allows to exchange individual layers (e.g. to follow technological progress) without affecting other layers as long as they keep their inter-layer interfaces. Figure 2.1 shows the seven layers of the ISO/OSI model on the left.

The vertical communication between layers is governed by interfaces. The main task of an interface in this context is data encapsulation: If a message from one layer N is passed to the next lower layer $N-1$, it becomes the payload of this lower layer. The lower layer typically adds its own management data, e.g. a header and/or trailer, to this payload to create a layer $N-1$ message, which is then passed to the next lower layer until the lowest (physical) layer is reached. At the physical layer the message is actually transmitted to the receiving network device(s), where the encapsulation process is reversed.

Horizontal communication between corresponding layers is governed via protocols. A protocol is a well-defined set of rules to allow data exchange between two or more communication entities. With the exception of the physical layer, these entities never communicate directly.

Next, we give a brief introduction to the layers of the ISO/OSI model with a focus on Ethernet and commonly-used higher-layer protocols.

2.1.1.1 Physical Layer

The physical layer defines the communication medium and the physical network topology. It is responsible for the encoding, signaling, and transmission of raw data between network interfaces. The chips that implement physical layer functions are often called PHY.

Ethernet's physical layer is split into several sublayers, as shown in Figure 2.1 on the right. The lowest of these sublayers is the actual communication medium. The next sublayer is responsible for physical signaling, i.e. the actual signaling and encoding for the communication medium. It communicates with the medium via a medium dependent interface (MDI) and with the next-higher sublayer, the reconciliation sublayer, with a standardized medium independent interface (MII).[1] The reconciliation sublayer translates between the MII and the lowest sublayer of ISO/OSI data link layer, the medium access control (MAC) layer.

A brief overview of physical layers for automotive Ethernet is given in Section 2.1.2.

2.1.1.2 Data Link Layer

The data link layer is responsible to allow end station to end station communication. To this end, it introduces (physical) addressing and defines message formats (cf. Sections 2.1.3 and 2.1.4). It it also responsible to arbitrate the access to the physical layer.

In Ethernet, the data link layer is split into three sublayers, two of which are optional (see Figure 2.1). The MAC sublayer implements the logical Ethernet functions such as addressing, framing, error detection, and arbitration to the physical layer (link arbitration). The MAC control sublayer (if present) implements features, which are signaled by special messages called MAC control frames. One example of such a feature is simple flow control via PAUSE frames [111], [84].[2] The logical link control (LLC) sublayer is responsible to harmonize various medium access control layers towards the network layer.

We will continue the discussion of Ethernet's data link layer later in this chapter.

2.1.1.3 Network Layer

The network layer enables (inter-)networking independent of a network's actual layer 1 and 2 implementations. It introduces logical addresses for network interfaces and means to translate them to layer 2 addresses. Its independence from the lower layers allows the network layer for example (1) to interconnect networks of different technologies by routing messages between them and (2) to transport messages independent of lower layer constraints. The latter point is enabled by automatic message fragmentation and reassembly transparent

[1] Different MII standards evolved over time (e.g. GMII, RMII, RGMII, SGMII) addressing the needs of faster physical layers and the reduction of production costs.

[2] Note that PAUSE frames block all traffic on a port for a specified amount of time [111]. While priority-based flow control allows to block only a certain traffic class (priority), it sill is a very coarse-grain means of flow control [84].

to higher layers. The internet protocol (IP) [89], [42] is an example of a typical network layer protocol.

2.1.1.4 Transport Layer

The transport layer enables end-to-end communication between processes running on end stations. It introduces additional addresses (often called ports) to distinguish between different processes. Often, transport layer protocols implement advanced features such as (automatic) message segmentation, e.g. to allow continuous data streaming over message-based communication or connection-based communication, including connection establishment, reliable message transport, and flow control. The user datagram protocol (UDP) [141] and the transmission control protocol (TCP) [90] are typical examples of transport layer protocols.

2.1.1.5 Session, Presentation, and Application Layer

The session, presentation, and application layers are application-oriented and are responsible to enable the implementation of applications running on a network. Their responsibilities include, for example, user-transparent connection control (checkpointing and restarting), compression and encryption, and machine-independent data representation.

2.1.2 Physical Layers for Automotive Ethernet

Due to different operating conditions an cost constraints, automotive Ethernet has different requirements regarding the physical layer than, for example, data center, office, and consumer domains. The key requirements in the automotive context are: (1) Low cost and mechanically robust cabling, preferably unshielded twisted pair (UTP) cables along with standard automotive connectors. (2) Automotive EMC requirements must be met, i.e. the PHY must operate without being affected by the electromagnetic emissions of other devices while also not disturbing the operation of other devices by its own emissions. This is especially important when UTP cabling is used. (3) Support cable lengths of at least 15 m with up to four inline connectors (to met manufacturing and maintenance requirements). (4) Operate in an extended temperature range of typically -40°C to 125°C.

These requirements lead to the development of new automotive compliant physical layer standards, namely BroadR-Reach [29], IEEE 802.3bw [4], and IEEE 802.3bp [2]. The former, BroadR-Reach, has been standardized by the OPEN Alliance [137], while the latter two are standardized within IEEE 802.3 (like most other Ethernet standards).

2.1.2.1 BroadR-Reach

The BroadR-Reach PHY [29] is standardized by the OPEN Alliance [137]. It specifically addresses the aforementioned automotive requirements and

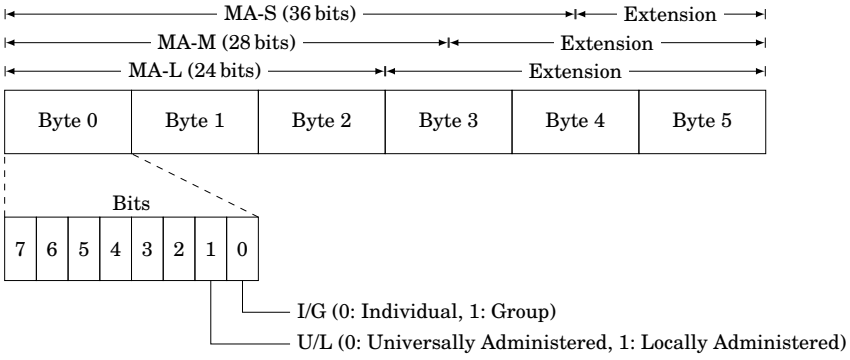


Figure 2.2: IEEE 802 48 bit MAC address format

provides a data rate of 100 MBit/s over a single pair of UTP cable. BroadR-Reach leverages implementation details and ideas from the IEEE 802.3ab gigabit Ethernet standard [73].

2.1.2.2 IEEE 802bw: 100 MBit/s over a Single Twisted Pair Cable

IEEE 802.3bw (100BASE-T1) [4] is an effort to standardize BroadR-Reach as part of IEEE 802.3 to ensure interoperability with the rest of the IEEE 802 family of standards.

2.1.2.3 IEEE 802bp: 1 GBit/s over Single Twisted Pair Cable

Many automotive use cases, e.g. multi-camera systems, require gigabit data rates. The main properties of a gigabit interconnect are: (1) the link delays are reduced by a factor of 10 compared to 100 MBit/s and (2) the increased bandwidth. IEEE 802.3bp [2] (1000BASE-T1) addresses these requirements and standardizes an Ethernet physical layer capable of operating at a data rate of 1 GBit/s over a single unshielded twisted pair cable in automotive and industrial environments.

2.1.3 MAC Addresses

In order to enable communication between different end stations, Ethernet requires an addressing mechanism. Media access control (MAC) addresses are used to unambiguously identify network interfaces. They are formally defined by IEEE 802 [113]. Ethernet MAC addresses are 48 bits long and are divided into six bytes as shown in Figure 2.2. MAC addresses can be either universally administered or locally administered. Also, they may refer to an individual network interface or a group of interfaces.

2.1.3.1 Universally and Locally Administered MAC Addresses

Universally administered MAC addresses must be globally unique. To ensure this uniqueness, the IEEE established a hierarchical assignment scheme. In this scheme, the IEEE registration authority assigns MAC addresses in blocks to assignees. These MAC address blocks (MA) are offered in three different sizes: MA-L (2^{24} addresses), MA-M (2^{20} addresses), and MA-S (2^{12} addresses) (see Figure 2.2). Depending on the block size, the first 24, 28, or 36 bits are used to identify the block. The assignee (typically a manufacturer) can then generate unique MAC addresses for its network interfaces by assigning the extension bits within its block.

Locally administered MAC addresses are usually assigned by a network administrator and must only be unique within their layer 2 network. Use cases for locally administered MAC addresses include encoding serial numbers or (physical) location (e.g. to help debugging). These addresses can also be used to encode MAC addresses in a way to avoid hash collisions in the forwarding tables of switches.

The U/L (universal/local) bit of a MAC address is used to distinguish between universally and locally administered MAC addresses (see Figure 2.2). If this bit is 0, the address is universally administered and if it is 1, the address is locally administered. Note that the U/L is 0 in all addresses blocks assigned by the IEEE registration authority.

2.1.3.2 Unicast, Multicast, and Broadcast MAC Addresses

In each Ethernet network each network interface has a unique individual MAC address. Additionally, there are special MAC addresses, which do not correspond to individual network interfaces, but are used to address a group of network interfaces instead. These addresses are used to implement multicast and broadcast communication. Multicast groups are used to deliver messages to a selected group of end stations. Broadcast messages are delivered to all end stations. If switches are used, they must have sufficient information to forward multicast traffic accordingly.[3] Broadcast traffic is usually flooded to all switch ports.

The I/G (individual/group) bit of a MAC address is used to distinguish between individual (unicast) and multicast/broadcast addresses (see Figure 2.2). If this bit is 0, the address is an individual address and if it is 1, the address is either a multicast or broadcast address. For Ethernet, the broadcast address is defined to contain all 1s [74].

2.1.4 Ethernet Frame Format

A frame format defines the structure of a message and, consequently, is a fundamental mechanism to enable communication. IEEE 802.3 [74] defines Ethernet frame formats at the MAC sublayer. In this thesis we focus on the frame format introduced by the DIX (Digital Equipment Corporation, Intel,

[3] This forwarding information is ether leaned from monitoring higher-layer protocols (e.g. IGMP snooping) or must be programmed statically.

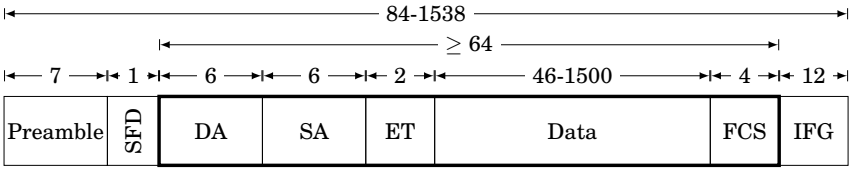


Figure 2.3: Ethernet frame (Ethernet II format). SFD: start of frame delimiter, DA: destination address, SA: source address, ET: EtherType, FCS: frame check sequence, IFG: interframe gap. All units are given in bytes.

Xerox) Ethernet standard version 2 (Ethernet II) frame. Its format is illustrated in Figure 2.3. Note that, in IEEE 802.3, the Ethernet frame is defined to include all fields from destination address to frame check sequence (highlighted fields in Figure 2.3), while term Ethernet packet is used to refer to the Ethernet frame plus preamble and start of frame delimiter. We, however, will use the term Ethernet frame to refer the entire structure shown in Figure 2.3, i.e. our definition includes all fields form the preamble to (and including) the interframe gap. The reason for this is twofold. First, during a worst-case timing analysis, e.g. when computing blocking times, an Ethernet frame is also delayed during the interframe gap (cf. Section 4.2.1). Second, to avoid confusion with IP packets, we will use the term frame.

Ethernet frames have a minimum and maximum length constraint. The minimum payload size is 46 bytes, resulting in a minimum frame size of 64 bytes according to the IEEE 802.3 Ethernet frame definition and 84 bytes according to our definition. This minimum frame size constraint must be enforced by padding the data field if necessary.[4] The maximum frame payload length is 1500 bytes, resulting in a maximum overall frame length of 1538 bytes (according to our definition).

2.1.4.1 Ethernet Frame Fields

Here, we briefly present the individual fields of an Ethernet frame (as shown in Figure 2.3).

2.1.4.1.1 Preamble

The preamble is a sequence of 56 alternating 1s and 0s (7 bytes). It was used by older PHYs to signal the start of a transmission and to give the receiver enough time to synchronize its clock to the starting transmission. Modern PHYs, typically maintain a synchronized point-to-point connection and technically do not require any preamble. It is kept, however, to maintain backwards compatibility.

[4]The reason for this constraint is that, in CSMA/CD implementations of Ethernet, a frame must be long enough to be able to propagate to every device on the network and back again in order to allow a sender to determine if a collision occurred before the frame transmission has ended.

2.1.4.1.2 Start of Frame Delimiter (SFD)

The start of frame delimiter is a special byte (0xd5) to signal the actual start of the frame after the preamble.

2.1.4.1.3 Destination Address (DA)

The destination address is a MAC address (see Section 2.1.3), which indicates the receiver(s) of an Ethernet frame. It may be an unicast (individual), multicast, or broadcast MAC address.

2.1.4.1.4 Source Address (SA)

The source address is a MAC address specifying the originating network interface of the Ethernet frame. Only unicast MAC addresses are allowed as source addresses.

2.1.4.1.5 EtherType (ET)

The EtherType is a mechanism to embed additional information into Ethernet frames on layer 2.[5] Use cases are the indication to which higher-layer protocol handler the payload data shall be passed, e.g. IPv4, IPv6, PTP, or frame tagging in order to embed priority and virtual network information (cf. Section 2.2). EtherTypes can be chained, so that a single Ethernet frame may contain multiple EtherTypes. Commonly used EtherTypes include, for example, the VLAN tag (0x8100), IPv4 (0x0800), and IPv6 (0x86dd).

2.1.4.1.6 Data

The data filed holds the Ethernet frame's payload. The payload of an Ethernet frame typically contains messages from higher-layer protocols (see Section 2.4). If the payload length is shorter than 46 bytes, it must be padded.

2.1.4.1.7 Fame Check Sequence (FCS)

The frame check sequence is a 32 bit cyclic redundancy check (CRC) to check the integrity of the (non-constant parts of the) Ethernet frame [74]. It is computed over destination address, source address, EtherType, and data (including possible padding).

2.1.4.1.8 Interframe Gap (IFG)

The interframe gap (also called interpaket gap (IPG)) is an artificial delay to give other lower Ethernet layers (including the medium) some time to recover

[5] In earlier Ethernet framing standards the two bytes representing the EtherType were used exclusively to encode the length of the Ethernet frame, while Ethernet II framing considered them as the EtherType. In order to allow interoperability, EtherTypes with a value smaller than 1536 bytes are considered length information, while values greater than or equal are considered to be Ethernet II EtherTypes.

from a transmission.[6] Akin to the preamble, this gap is often not required by modern Ethernet PHYs and is kept for compatibility reasons.

2.2 Quality of Service

As an automotive backbone network, Ethernet must accommodate traffic from a wide and increasing set of applications, some of which are more critical than others (e.g. time-critical safety-related functions vs. high-bandwidth infotainment traffic). In Ethernet, however, as defined in IEEE 802.3, all frames are treated equally when they are being forwarded in the network. Quality of Service (QoS) is a mechanism used when, for example, some traffic streams should be given precedence over other streams, often in order to meet non-functional requirements. We start with a definition of QoS and then discuss how it is addressed in Ethernet.

Definition 1 (Quality of Service)**.** *Quality of Service (QoS) is the ability of a network to provide certain service guarantees to selected traffic streams regardless of the behavior of other traffic streams.*

In order to support QoS a network must provide means to (1) mark (tag) traffic streams for later identification, (2) enforce QoS, and (3) configure and manage QoS requirements.

In Ethernet, marking is defined by IEEE 802.1Q (see Section 2.2.1). Ethernet also supports various QoS enforcement policies. For QoS configuration and management, IEEE 802.1Q defines the (generic) multiple registration protocol (MRP) to register attributes in a database in each port (of a switch). Ethernet AVB's and Ethernet TSN's stream reservation protocols [71], [87] are based on MRP.

In the following sections, we will present and discuss a selection of these policies.

2.2.1 IEEE 802.1Q: Bridges and Bridged Networks

IEEE 802.1Q addresses quality of service on layer 2 and introduces priorities and virtual local area networks (VLANs) to Ethernet. In order to work, it requires additional information to be present in Ethernet frames. As IEEE 802.1Q operates on layer 2, this information (also called tag) is embedded into Ethernet frames via the EtherType mechanism (cf. Section 2.1.4). Figure 2.4 shows an Ethernet frame with an embedded IEEE 802.1Q tag. Note that IEEE 802.1Q tagging reduces the maximum payload size of an Ethernet frame by 4 bytes.

2.2.1.1 IEEE 802.1Q Tag Fields

The IEEE 802.1Q tag has a size of 4 bytes. Its start is signaled by an EtherType with the value 0x8100. The tag's payload, i.e. its tag control information (TCI) contains the following fields.

[6] When Ethernet was used a shared medium, transceivers used this gap to switch from sending to receiving operation.

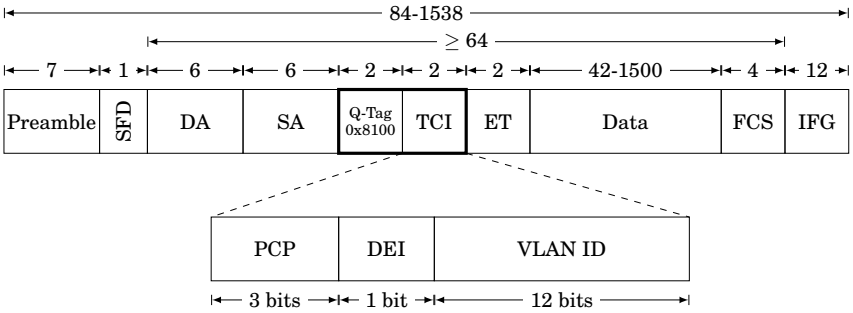


Figure 2.4: Ethernet frame with IEEE 802.1Q tag (cf. Figure 2.3). Q-Tag: IEEE 802.1Q tag, TCI: tag control information, PCP: priority code point, DEI: drop eligible indicator, VLAN ID: VLAN identifier. Dimensionless units are given in bytes.

2.2.1.1.1 Priority Code Point (PCP)

The priority code point is a 3 bit field to assign a priority to an Ethernet frame. Higher numbers correspond to higher priorities. Since the PCP field is 3 bits long, only eight priorities are supported. As most networks carry more than eight traffic streams, frames of different streams typically must share the limited number of priority levels. Consequently, the PCP provides a classification, rather than an identification mechanism.

2.2.1.1.2 Drop Eligible Indicator (DEI)

The drop eligible indicator is used to control how frames are dropped if a network device runs out of buffer space, e.g. due to congestion. A DEI set to 1 means that such frames should be dropped preferentially compared to those with a DEI set to 0. Note that setting the DEI to 0 does not prevent the frame from being dropped entirely.

2.2.1.1.3 Virtual Local Area Network Identifier (VLAN ID)

The VLAN ID is used to indicate to which virtual network a frame belongs (see Section 2.2.1.2).

2.2.1.2 Virtual Networks

Virtual local area networks (VLANs) are a mechanism to separate a single physical layer 2 network into different logical layer 2 networks. Technically, this is achieved by using the VLAN ID field in IEEE 802.1Q tagged frames (cf. Figure 2.4).[7] VLAN IDs can be assigned by (trusted) end stations or by switches

[7] VLANs can also be port-based, i.e. instead of adding an IEEE 802.1Q tag to a frame, switch ports are configured to serve exactly one VLAN. This approach, however, is less flexible as, for example, different VLANs cannot share the same switch port, which can increase the wiring effort if multiple

based on the port from which frames enter the network. For security reasons, switches typically also support to check VLAN IDs against a preconfigured set. The actual network separation is implemented by IEEE 802.1Q compliant switches that forward frames only to ports associated with matching VLAN IDs. As multiple VLANs can share the same link, a port may be associated with multiple VLANs. The main uses cases for virtual networks are (1) security, (2) performance, and (3) redundancy.

Network security can be enhanced by virtual networking if different VLANs are used to separate different security domains. As end stations might get compromised, VLAN IDs (or entire IEEE 802.1Q tags) are typically assigned or checked by switches. Note that this logical isolation is not necessarily of physical nature as, depending on the network configuration, different VLANs might still share the same link(s) inside a network and can potentially affect each other's timing behavior by interference.

Virtual networking can help to improve performance. First, by separating a (large) layer 2 network into several (smaller) virtual networks, the size of broadcast domains, and hence the amount of (potential) background traffic, is reduced. Second, certain traffic streams can be isolated by configuring VLANs such that these traffic streams never use the same links or even switches. This decouples their timing behavior by reducing or eliminating their mutual interference.

Ethernet requires a cycle-free layer 2 network to operate correctly. However, to compensate link or switch failures, redundancy, and hence, cycles (by adding extra links or switches) must be introduced. These cycles are typically detected and cut at layer 2 by the spanning tree, rapid spanning tree, or shortest path bridging protocols (STP, RSTP, SPB) [76]. The resulting cycle-free layer 2 topology is often called the active topology. In the case of a link or switch failure, these protocols are executed again to find and establish a new cycle-free layer 2 network topology. Consequently, as this configuration process takes time, they are not suitable for hot spatial redundancy according to automotive requirements. In this context, VLANs can be used to implement hot spatial redundancy by establishing independent cycle-free layer 2 networks on top of a non-cycle-free network. Note that with IEEE 802.1CB (Frame Replication and Elimination for Reliability) [78] and IEEE 802.1AX (Link Aggregation) [114] more elaborate means to provide spatial redundancy are available.[8]

2.2.1.3 Priority-Based QoS Enforcement

IEEE 802.1Q introduces a general model for bridge (i.e. switch) operation, including how frames are forwarded under QoS. The QoS enforcement in Ethernet is performed at the output ports. Each frame traverses three stages: (1) classification, (2) queueing, and (3) transmission selection. Figure 2.5 shows

VLANs are established over the same set of switches. An advantage of port-based VLANs is that there is potentially less interference between different VLANs, due to the absence of port sharing.

[8]While IEEE 802.1AX can only provide spatial redundancy between two topologically adjacent network devices (e.g. between two switches or a switch and an end station), IEEE 802.1CB allows the implementation of end-to-end spatial redundancy.

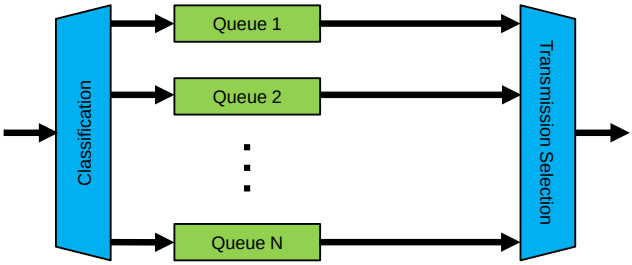


Figure 2.5: IEEE 802.1Q switch port model. Back arrows indicate possible frame data paths.

a switch port model illustrating these stages. This model will be used (and extended) throughout this section.

In the classification stage, frames are mapped to traffic classes based on their priority, i.e. their PCP field. Each traffic class is associated with a queue to hold frames awaiting transmission. A network interface of an end station or switch may implement less queues then there are priorities, e.g. for cost reasons. Consequently, in such cases multiple priority levels are mapped to the same traffic class and, hence, queue.

IEEE 802.1Q does not specify how queues should be implemented, apart from the constraint, that ordering of the frames belonging to the same traffic stream must be preserved. In this context, a traffic stream is only defined by layer 2 properties, i.e. frames with identical priority, VLAN ID, and destination and source MAC addresses are considered to belong to the same traffic stream. This allows a large degree of freedom when implementing these queues (as long as the ordering constraint of IEEE 802.1Q is met). Usually, however, these queues are implemented as FIFO buffers and this is also what we will assume in this thesis.

The transmission selection stage determines which frame should be transmitted next. IEEE 802.1Q requires that all switches shall support static priority based transmission selection.[9] When we compare different transmission selection schemes later, we will use IEEE 802.1Q as a synonym for static priority transmission selection. Other transmission selection schemes such as Ethernet AVB have been proposed in the context of the audio/video domain [72] or Ethernet TSN for time-sensitive applications [83], [80]. We will introduce various schemes in the following sections and discuss their performance from a formal worst-case timing perspective in Chapter 4.

Note that in Ethernet frame transmissions are typically non-preemptive, i.e. once a frame has started transmitting, it is guaranteed to finish its transmission without interference. Recently, however, preemption support has been added to Ethernet. We discuss frame preemption for Ethernet in Section 2.2.3.4 and present a formal worst-case analysis in Section 4.6.

[9] Static priority based transmission selection is also one of the EthSwtPortSchedulerAlgorithms of the AUTOSAR Ethernet switch driver module [38].

2.2.2 Ethernet AVB: Audio Video Bridging

Ethernet AVB is a set of standards addressing the quality of service requirements for audio and video (A/V) streaming in Ethernet networks. Its goal is to enable the streaming of time-sensitive A/V data, e.g. synchronized playback across multiple (spatially separated) speakers or lip-synchronous A/V playback. The original Ethernet AVB application domains include home entertainment and professional A/V (e.g. live performances or studio environments). However, its properties also make it a suitable candidate for automotive infotainment [123].

In order to achieve its goals, Ethernet AVB defines methods to provide time synchronization throughout the network, to provide bounded latency and jitter for certain traffic streams,[10] and to manage network resources (e.g. by resource reservation).

Ethernet AVB is specified across a set of standards, most of which are part of the IEEE 802 family. In the following sections, we present a brief overview of these standards.

2.2.2.1 IEEE 802.1AS: Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks

IEEE 802.1AS (also called the generalized precision time protocol (gPTP)) is a protocol to provide and maintain network-wide time synchronization for time-sensitive applications on IEEE 802 media. It is a simplified extension to IEEE 1588 [75] (precision time protocol (PTP)) and works solely on layer 2, whereas IEEE 1588 also supports layers 2 and above.

To establish network-wide time synchronization, one network device acts as a grandmaster clock and all other devices synchronize their clocks to this grandmaster. The grandmaster can either be preconfigured or determined at runtime by a distributed best master clock selection algorithm (BMCA). With BMCA, every potential grandmaster device broadcasts its clock characteristics to all other devices in the network and the device with the best clock is then selected to be the grandmaster. Whether a clock is better than another clock is determined based on properties such as clock accuracy and priority.

Time synchronization is established by the grand master by periodically sending a pair of sync and follow_up frames through the network. Sync frames contain a time stamp from the grand master's clock, while follow_up frames keep track of how long it took a sync frame to propagate from the grand master to an end station (this includes measuring of how long a frame was queued as well as the link delay between Ethernet devices). With this information an end station can compute the actual time. IEEE 802.1AS operates on a device to device level, i.e. a device only talks to its immediate neighbors, which then have to talk to their neighbors.

Automotive networks generally have very tight timing requirements regarding their start-up process. Consequently, the grandmaster will very likely be preconfigured [123]. Additionally, [123] suggests to store the (last) learned

[10] Ethernet AVB, particularly IEEE 802.1Qav, can only give certain bandwidth guarantees. No formal latency guarantees for time-critical traffic can be given (see Section 2.2.2.2 and [123]).

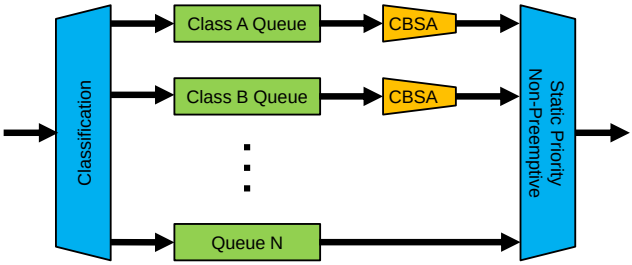


Figure 2.6: IEEE 802.1Qav port model. Note that Class A does not necessarily have to be on the highest priority.

wire delays to speed up the start-up synchronization. This is possible as the in-vehicle network architecture and wire lengths rarely change.

2.2.2.2 IEEE 802.1Qav: Forwarding and Queuing Enhancements for Time-Sensitive Streams

IEEE 802.1Qav [72] defines Ethernet AVB's QoS enforcement mechanism. Similar to IEEE 802.1Q, traffic streams in Ethernet AVB are mapped to classes based on their priority levels. The AVB classes (also called stream reservation (SR) classes) are identified by consecutive letters, starting with class A. Classes with letters closer to A typically are assigned to a higher priority. There can be up to seven AVB classes. As Ethernet (IEEE 802.1Q) only supports eight traffic classes, at least one class is left for non-AVB traffic. Each AVB traffic class is associated with a bandwidth reservation (cf. Section 2.2.2.3).

IEEE 802.1Qav's design goal is to distribute the frames of AVB traffic classes evenly over time according to their bandwidth reservation. The motivation for this is that unevenly distributed traffic (with the same bandwidth), e.g. a large continuous number of frames (burst) followed by a long idle time, from one stream typically blocks individual frames from other streams longer as if these frames had been distributed more evenly, e.g. with (small) idle times between consecutive frames. To this end, IEEE 802.1Qav specifies traffic shaping in the form of a so-called credit-based shaper algorithm (CBSA). CBSA is located between the queues and transmission selection in the AVB switch port model as illustrated in Figure 2.6. Each queue associated with an AVB traffic class has its own CBSA shaper. The final transmission selection in IEEE 802.1Qav is static priority non-preemptive (SPNP).

CBSA works by limiting the number of frames that are allowed to pass the shaper based on the bandwidth reservation of the AVB traffic class associated with it. The shaper's state is determined by a credit counter. Frames are allowed to pass the shaper only if the credit is greater than of equal to 0. The credit is decremented by a sendSlope while a frame is being transmitted. If the credit becomes negative during a transmission, the on-going transmission is allowed to finish, but subsequent transmission are blocked until credit replenished to at least 0. The credit is incremented by an idleSlope whenever a frame's

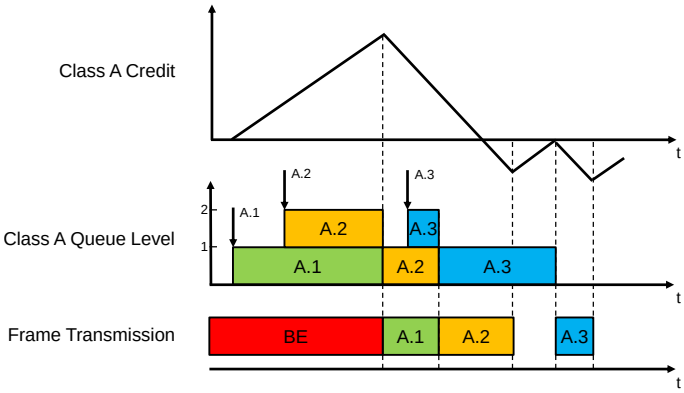


Figure 2.7: CBSA example for a class A queue that is initially blocked by a best effort (BE) frame and then releases its frames in a shaped manner, starting with a small burst to catch up on its bandwidth reservation. The class A credit graph shows class A's credit over time, the class A queue level graph shows the frames (arrows represent frame arrivals) queued (stacked on the x axis) in class A's queue over time (i.e. boxes of the same color do not represent the frame length but their residence time in the queue), and the frame transmission graph shows the actual frame transmissions on the wire to the next device.

transmission is blocked by interfering traffic or when the credit is negative. CBSA allows to specify minimum and maximum credit bounds. idleSlope and sendSlope reflect the bandwidth reservation of an AVB traffic class.

Figure 2.7 illustrates the operation of CBSA for a single AVB traffic class. Note that without any blocking CBSA emits frames evenly according to their reserved bandwidth. It allows bursts only to enable an AVB traffic class to catch up on its bandwidth reservation after it has been blocked by interfering traffic.

As discussed, in Ethernet typically multiple different traffic streams have to share a single priority and, hence, an AVB class. Ethernet AVB addresses the potential intra-class interference of multiple traffic streams by also requiring per traffic stream CBSA traffic shapers at the end stations.

2.2.2.3 IEEE 802.1Qat: Stream Reservation Protocol (SRP)

Quality of service guarantees, such as the ones given by IEEE 802.1Qav rely on the availability of sufficient resources on the path between a sender and receiver. IEEE 802.1Qat [71] describes a decentralized admission control and resource reservation protocol for traffic streams, the stream reservation protocol (SRP). It ensures that sufficient bandwidth is available between communication partners before their actual data exchange begins. If the available bandwidth is insufficient, the communication is denied.

SRP is based on MRP (cf. Section 2.2). A sender advertises the availability of streaming data in the entire network via MRP talker advertise attributes and a

receiver can subscribe to such data streams via listener ready MRP attributes. The switches on the path between sender and receiver check the attributes and make the corresponding reservations or, in the case of insufficient resources, signal failure via MRP.

From an automotive perspective, run-time stream reservation might be too time consuming. Furthermore, SRP is not allowed to fail for safety-critical traffic. One proposed solution is to preconfigure all resource reservations, e.g. by running SRP once at the end of the assembly line and permanently store the result or to hard code all resource reservations during the design process [123].

2.2.2.4 IEEE 802.1BA: Audio Video Bridging (AVB) Systems

The IEEE 802.1BA [70] standard defines profiles and corresponding profile conformance statements to help manufacturers to develop AVB-compatible products and to enable end users to design networks capable of transporting time-sensitive A/V traffic based on these profiles.

2.2.3 Ethernet TSN: Time-Sensitive Networking

Ethernet AVB has been designed to meet the requirements of audio and video applications. It, however, does not cover the rigid requirements in the automotive and industrial control domains, including ultra-low latencies and various safety requirements. After the end of the Ethernet AVB standardization process, a new standardization effort under the more general name of Ethernet time-sensitive networking (TSN) [85] was started to address these requirements.

Ethernet TSN, like Ethernet AVB, is specified across a set of standards within the IEEE 802 family. Its main design goals are network-wide time synchronization of end stations and switches, quality of service (e.g. bounded latency, zero congestion loss, and increased reliability), ingress filtering and policing to protect the network from rogue traffic streams and to facilitate fault containment, frame preemption to reduce blocking delays, support for scheduled traffic (e.g. TDMA), and explicit path configuration. Additionally, Ethernet TSN includes standards to manage and configure all these features.

In the following sections, we present an overview of these standards.[11] We will introduce the ones relevant to this thesis (i.e. IEEE 802.1Qbv, IEEE 802.1Qch, IEEE 802.1Qbu) in more detail and, for the sake of completeness, briefly present the others.

2.2.3.1 IEEE 802.1AS-Rev: Timing and Synchronization for Time-Sensitive Applications

IEEE 802.1AS-Rev [77] is a revision of IEEE 802.1AS addressing the requirements of time-sensitive networking. Its key features over IEEE 802.1AS include improved grandmaster redundancy and faster synchronization.

[11] Note that some standards are still in their draft phase and only limited or preliminary information is available.

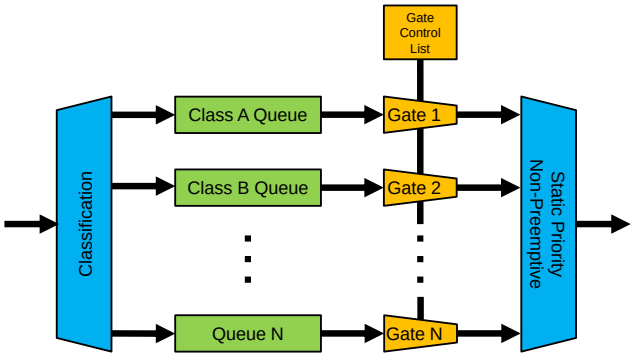


Figure 2.8: IEEE 802.1Qbv port model

In IEEE 802.1AS, when a grandmaster clock fails, there are two alternatives: (1) If BMCA is enabled, then a new grandmaster clock is selected via BMCA. (2) If the grandmaster clock was preconfigured, then there will be no more clock synchronization. Both alternatives are typically undesirable in the automotive context. In order to support immediate failover, IEEE 802.1AS-Rev allows the definition of multiple (overlapping and concurrently active) time domains, each with its own grandmaster clock.

To achieve faster synchronization, IEEE 802.1AS-Rev reduces the two-step synchronization process from IEEE 802.1AS (cf. Section 2.2.2.1) to one step by allowing the embedding of time stamps into sync frames on-the-fly.

2.2.3.2 IEEE 802.1Qbv: Enhancements for Scheduled Traffic

IEEE 802.1Qbv [83] (also called time-aware shaper (TAS)) adds support for scheduled traffic, i.e. sending frames at predefined points in time, by defining a network-wide traffic-class-based TDMA scheme. TDMA is typically used when detailed temporal information about data production and consumption is available. This is often the case for control applications, which rely on periodic sampling and control. By design, TDMA schemes also allow temporal isolation of different traffic classes. A drawback of TDMA schemes, however, is that they often are inflexible and inefficient regarding the accommodation of non time-triggered traffic.

IEEE 802.1Qbv is implemented by adding per-class transmission gates to the output port model of Ethernet (see Figure 2.8). A transmission gate disconnects a traffic class' queue from participating in transmission selection when closed, and (re-)connects the queue to transmission selection when opened. The times at which these gates open and close, i.e. the time intervals at which each traffic class can schedule its frame transmissions, are programmable via a gate control list. The transmission selection scheme in IEEE 802.1Qbv is static priority non-preemptive (SPNP). If the transmission gates of multiple traffic classes are open concurrently, link access is arbitrated according to the priority of these classes.

Due to non-preemptive transmission selection, a frame that started transmitting just before the gate of its traffic class closes would overlap into the scheduled interval of another class and would delay its scheduled frame transmissions. The delay from such an overlap would be significant, e.g. up to 120 us for a maximum-sized frame at 100 MBit/s. In order to prevent this and enable a traffic class carrying scheduled traffic to transmit its frames at predefined points in time, IEEE 802.1Qbv introduces so called guard bands. Guard bands are time intervals, which are inserted before the gate of a traffic class closes and during which no frame transmission of that class is allowed to start. IEEE 802.1Qbv constrains frame transmissions such that a frame is only allowed to start transmitting if the gate associated with its traffic class is open and if it will finish its transmission before its class' gate closes. Consequently, these guard bands must be sufficiently large, which can result in poor link utilization. Another source of poor link utilization is that leftover bandwidth, i.e. bandwidth that is not used in a scheduled interval of one class cannot, be reused by other classes. Note that, as IEEE 802.1Qbv defines a per-class gate schedule, the frames within a given scheduled traffic class still concur for transmission selection.[12]

As a TDMA scheme, IEEE 802.1Qbv can give very low latency and jitter guarantees if all involved components are fully synchronized. While this can be achieved with IEEE 802.1Qbv with the support of IEEE 802.1AS (see Section 2.2.2.1) in Ethernet networks, full system-wide synchronization often cannot be ensured due to different communication paradigms. Particularly, whenever messages are passed from an event-triggered domain into a time-triggered domain, there is a certain sampling delay. This is the time a message has to wait for its TDMA slot under worst-case circumstances. In-vehicle communication architectures are a prominent example. Here, the CAN bus is event-triggered and scheduled via SPNP. Consequently, an inter-domain message, that uses an IEEE 802.1Qbv Ethernet backbone, could suffer from sampling delay [172].

Furthermore, their static nature makes TDMA schemes inflexible to changes of the system architecture. For example, the integration of new end stations or functions might require the design of a new TDMA schedule. In Ethernet networks this is a complex task, as the TDMA schedules across multiple end stations and switches must be carefully synchronized to each other.

2.2.3.3 IEEE 802.1Qch: Cyclic Queuing and Forwarding

In priority-based transmission selection schemes, the end-to-end delay along a path through a network typically depends on the interfering traffic, which might not be known entirely at design time. The key idea of IEEE 802.1Qch [80], [165] is to bound the (maximum) residence time at each switch so that the determination of the end-to-end delays only depends on the number of switches on the path between sender and receiver.

In IEEE 802.1Qch, frames are not only classified based on their priority, but also based on their arrival time. Time is divided (network wide) into alternating

[12] Frames of the same traffic class are typically FIFO-scheduled in the queue associated with their class (cf. Section 2.2.1.3).

cyclic intervals of equal length (even, odd). At each switch, frames received in an even interval are scheduled to be transmitted in the next odd interval and vice versa. The cyclic intervals must be large enough to transmit all frames that can arrive during an interval.[13] Then, the delay a single frame can experience while being forwarded through a switch is upper bounded by twice the cyclic interval length (in the worst-case the frame arrives as the last one in a burst at the beginning of its reception interval and is then send out last in the following transmission interval).

However, if the cyclic interval is too short, i.e. more frames need to be transmitted than time is available in the next interval (e.g. due to a transient overload), an appropriate action is required. Such actions include dropping the excess frame(s) or reducing their priority [80]. Another approach is to keep transmitting the excess frames [165]. In this case the transmission of frames that are scheduled to be transmitted in subsequent intervals is delayed until the overload has been processed.

2.2.3.4 IEEE 802.1Qbu and IEEE 802.3br: Frame Preemption and Interspersing Express Traffic

Traditionally, frame transmission in Ethernet has been non-preemptive, regardless of the arbitration mechanism. Under non-preemptive frame transmission, a frame, which is in transmission, is guaranteed to finish without interruption. Hence, a time-critical high-priority frame can be delayed by a frame of lower-priority, if the high-priority frame arrives during the transmission of the lower-priority one. For instance, in the worst-case, high-priority frames can be delayed by about 120 us per switch at 100 MBit/s links by lower-priority frames.

The IEEE 802.3br (interspersing express traffic) standard [3] addresses this problem by introducing frame preemption to Ethernet. In the context of Ethernet TSN, it is supported by IEEE 802.1Qbu (frame preemption) [79], which adds management and configuration mechanisms for frame preemption. IEEE 802.3br only allows one level of preemption. To this end, it defines two MAC interfaces: the express MAC interface and the preemptable MAC interface (see Figure 2.9). Each Ethernet traffic class is mapped to either the express or the preemptable MAC interface. Frames of express classes cannot be preempted. Only frames of classes which are mapped to the preemptable MAC interface may be preempted by express frames (into smaller fragments). Particularly, preemptable frames cannot be preempted by other preemptable frames regardless of their priority. Note that IEEE 802.3br defines link-level frame preemption, i.e. frames are split into fragments and are reassembled at the MAC interfaces, so that end stations and switches (internally) only process complete frames.

With frame preemption the maximum delay that can be experienced by a higher-priority frame due to lower-priority blocking is reduced significantly to about 12 us per switch at 100 MBit/s. Frame preemption, however, also causes a certain overhead for preempted frames, which can have a measurable

[13] We will argue in Section 4.5 that choosing a sufficient cyclic interval length still depends on interfering traffic (or a conservative estimation of this traffic).

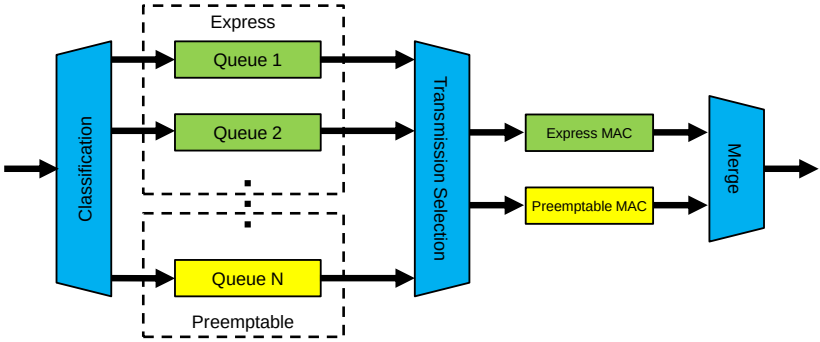


Figure 2.9: Port model of IEEE 802.3br with express and preemptive MAC interfaces

negative performance impact. This can become a problem when critical traffic streams with large frames, which are not scheduled on the highest priority, are preempted. ADAS (camera) traffic, which is typically scheduled on a priority below high-priority control traffic, is a typical example.

2.2.3.5 IEEE 802.1Qcr: Asynchronous Traffic Shaping

Burst control is an important goal in time-sensitive networking. One adverse effect (known as burstiness cascade [115]) is that the burstiness of an individual traffic stream within a class of traffic streams may lead to increased burstiness of other traffic streams along its path. As Ethernet AVB's CBS operates on traffic classes, it is, in consequence, traffic stream agnostic and is only of limited use to address this problem. The ideal mechanism to solve this issue would be per-stream (re)shaping (cf. [19]), but this is considered to be uneconomical with current hardware [159]. Instead, IEEE 802.1Qcr [82] proposes an interleaved shaping mechanism, called asynchronous traffic shaping.

The key principle of the asynchronous traffic shaper is the separation of buffering and shaper state. Instead of implementing many queues (e.g. one for each traffic stream) and managing them during shaping, IEEE 802.1Qcr operates on a limited number of shaped FIFO queues and only maintains per-stream shaper states. More concretely, IEEE 802.1Qcr uses two classes of FIFO queues: shaped and shared queues. Shaped queues are used to buffer frames that are waiting to be shaped. Frames are enqueued to shaped queues according to a set of rules (based on their origin and priority) to enforce sufficient isolation between (predecessor) devices (e.g. to prevent unexpected (over)load) and to be able to process the shaped queues in pure FIFO order. The frames at the heads of the shaped queues are compared to their shaper state, i.e. whether they are eligible for transmission according to a preconfigured (parameterized) traffic pattern, and, if they conform to this traffic pattern, they are enqueued into a shared queue. Shared queues represent the normal per-class queues of a switch's output port.

The shaping (i.e. the decision when a frame is eligible to be enqueued into a shared queue) is done on a local clock basis (e.g. the device clock) and was intended to not require synchronization to a network-wide timebase (hence the shaper name).[14]

2.2.3.6 IEEE 802.1Qci: Per-Stream Filtering and Policing

Safety critical applications such as ADAS and autonomous driving require fault tolerance mechanisms. From a network perspective, faults can manifest themselves in various ways. A common manifestation is that a device turns into a babbling idiot, i.e. it sends unusual high amounts of traffic into the network (e.g. by continuously broadcasting (maximum-sized) frames).[15] The babbling might be limited to a single traffic stream originating from the device, but could also very well affect multiple or all traffic streams from that device. Either way, babbling results in faulty traffic streams, which can have a negative impact on the timing behavior of other traffic streams, due to increased blocking and interference.

The IEEE 802.1Qci standard addresses the fault containment aspect of fault tolerance, i.e. it stops faulty traffic streams from propagating, by specifying per-stream filtering and policing mechanisms. While per-class filtering and policing would require less hardware (there are at maximum 8 traffic classes in Ethernet (see Section 2.2.1)), it would not provide adequate fault containment, as a traffic class potentially comprises multiple independent traffic streams (which would all be blocked by per-class filtering and policing).

2.2.3.7 IEEE 802.1CB: Frame Replication and Elimination for Reliability

In the automotive and industrial domains, reliable frame transmission is required to meet certain safety requirements (e.g. fail operational behavior). The main design goal of IEEE 802.1CB [78] is to achieve a high probability of frame delivery. It targets networks, where the time required to reconfigure the network via dynamic network control protocols (e.g. SDN [107]) in case of device or link failures is unacceptable. To this end IEEE 802.1CB defines hot spatial redundancy for Ethernet networks, which is realized by the redundant transmission of Ethernet frames over different (independent) paths.

In particular, it defines mechanisms to create multiple redundant traffic streams via the introduction of stream identifiers and sequence numbers (e.g. via a redundancy tag) and to recover the original traffic stream by detection and removal of duplicated frames. The recovery mechanism is also able to detect latent errors, i.e. errors (such as link loss) that have been masked by redundancy. The mechanisms of IEEE 802.1CB can be applied to both end stations and switches.

[14] In practice, however, a common network-wide understanding of shaped bandwidth is required, which, ultimately, depends on a common understanding of time. For the asynchronous traffic shaper, in particular, additional means are required to guarantee deterministic communication [174].

[15] Attacks can also result in unusual traffic patterns, which can potentially also be detected and handled by IEEE802.1Qci.

In order to achieve its goal of reliable frame transmission, IEEE 802.1CB must be configured tightly to the environment it operates in, carefully considering both the network topology as well as the network traffic (including interfering traffic) [143]. Furthermore, IEEE 802.1CB deliberately breaks with the IEEE 802.1Q requirement, that ordering of the frames belonging to the same traffic stream must be preserved (cf. Section 2.2.1.3), which may require corresponding reordering support at the higher protocol layers.

2.2.3.8 IEEE 802.1Qcc: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements

Ethernet TSN introduces various QoS mechanisms (e.g. scheduled traffic, cyclic forwarding, frame preemption, filtering and policing, redundancy), all of which require some degree of configuration before they can be used. The design goal of IEEE 802.1Qcc [81] is to add support for these mechanisms to SRP, which has originally been specified in the Ethernet AVB context (see Section 2.2.2.3).

It also introduces support for centralized network configuration, where users can state their communication requirements. Network configuration can be performed by three models: (1) the fully distributed model, (2) the centralized network/distributed user model, and (3) the fully centralized model. These models differ mainly in the flow of configuration information in the network.

2.2.3.9 IEEE 802.1Qca: Path Control and Reservation

Typically, (R)STP [76] or SPB [112] are used to derive a single active, i.e. cycle-free, topology in Ethernet networks. Safety-critical applications, however, often require multiple active topologies, e.g. to implement redundancy, to reduce congestion, or to enforce isolation of critical and non-critical traffic streams. IEEE 802.1Qca addresses these use cases and defines mechanisms to (explicitly) compute, configure, and manage different paths and active topologies.

One of the main features of IEEE 802.1Qca is the introduction of a path computation element (PCE) [55]. A PCE is an entity that is capable of computing network paths based on the network topology and a set of constraints, including preconfigured paths.

IEEE 802.1Qca can be used to provide (redundant) paths for IEEE 802.1CB along with IEEE 802.1Qcc, which can make corresponding resource reservations.

2.2.3.10 IEEE P802.1DG: Time-Sensitive Networking Profile for Automotive In-Vehicle Ethernet Communications

At the time of writing, IEEE P802.1DG [110] is an actively developed standardization effort for automotive Ethernet TSN profiles. It discusses the context and typical scope of automotive Ethernet TSN networks and defines profiles to streamline the requirements specification by manufacturers and implementation by suppliers, and also to help users to quickly assess functionality and interoperability, as well as aiding (protocol) testers with validation.

Currently, there are two profiles defined: (1) a base profile and (2) an extended profile. The base profile addresses requirements of infotainment and

ADAS systems as well as security requirements by requiring time synchronization (IEEE 802.1AS), credit-based shaping (IEEE 802.1Qav), and basic ingress checks (IEEE 802.1Qci), respectively. The extended profile, additionally, addresses requirements for fault tolerance and deterministic latency. It requires, in particular, time synchronization with quick recovery times (cf. IEEE 802.1AS-Rev multi-domain support) and IEEE 802.1CB, and considers IEEE 802.1Qbv, IEEE 802.1Qch, and IEEE 802.1Qcr to address more demanding timing requirements, as well as frame preemption (IEEE 802.3br/IEEE 802.1Qbu).

It is worth noting that IEEE P802.1DG also includes informative discussions on the intricacies of IEEE 801.1AS, IEEE 802.1CB, and the various transmission selection mechanisms that are available in Ethernet TSN with a particular focus on the subtle interactions of the latter.

2.2.4 Weighted Round Robin and Deficit Round Robin

Weighted round robin (WRR) scheduling [102] is a round-based QoS mechanism, aiming to distribute the available bandwidth proportionally among concurring traffic classes. Formally, WRR is a low-complexity approximation of weighted fair queuing (WFQ) [44], [26]. In contrast to (normal, non-weighted) round robin scheduling, WRR uses weights to differentiate the service each traffic class receives in each round.

While (unshaped) priority-based transmission selection schemes (such as IEEE 802.1Q) can lead to long delays or starvation of lower-priority traffic classes, when higher-priority classes transmit continuously, WRR prevents this by giving each queue the chance to transmit a certain number of frames (depending on the weight) per round. WRR is available in many commercial Ethernet switches, including switches targeting the automotive domain.

Transmission selection in WRR is round-based. At the beginning of each round, each traffic class is assigned a certain (preconfigured) number of frames or bytes that it can maximally transmit during the round. This number is called the weight of the class. In each round, the scheduler visits all queues one after another. When a queue is visited, the scheduler starts transmitting frames from that queue until its weight has been reached or until the queue is empty, whichever occurs first. Empty queues are skipped. Each queue is typically visited only once per round and leftover weight cannot be carried over into the next round. Frames arriving after their associated queue has been served must wait until the next round.

Deficit round robin (DRR) [158] is another approximation of WFQ, which typically implements byte-based weights. In contrast to WRR, in DRR the unused (remaining) weight of a non-empty queue, e.g. if the remaining weight is smaller than the frame at the head of the queue, can be carried over into the next round. This improves the fairness of DRR.

2.3 Ethernet Switches

Switches forward Ethernet frames towards their destination. Consequently, they are the key components of Ethernet's communication infrastructure.

Safety-critical real-time systems require freedom from interference [93] or sufficient independence [91] between critical and non-critical components (including their communication). In Ethernet, these requirements are addressed by various QoS mechanisms, many of which require support in the switches (see Section 2.2). While the IEEE 802.1 standard family specifies QoS mechanisms to ensure interoperability between different switch implementations, it deliberately does not provide any specific implementation details. However, a switch implementation can introduce complex interdependencies between traffic streams on various levels and, hence, threaten freedom from interference outside the reach of the IEEE specification. In the automotive context, the OPEN Alliance defines a minimal feature set for automotive switches [138].

Ethernet switch implementations are typically a trade-off. They provide a certain degree of functionality while being constrained by limited resources, due to, for example, cost or power budgets. Frame buffering is an example. Switches only contain a finite amount of memory, which can only store a finite amount of frames during times of congestion. If the memory is full while new frames arrive, frame drop is inevitable (cf. Section 2.2.1.1.2). For safety-critical real-time systems, however, it must be ensured that, regardless of the trade-off, QoS mechanisms still work as intended, e.g. in our example a formal determination of the worst-case switch memory utilization, as described in Section 4.3.4, is required.

In order to get a sound understanding of the impact of the switch architecture on QoS and safety, a detailed model of the switch architecture is required. While the concrete switch architecture is highly vendor-specific, many switches share common high-level design concepts. In the following, we introduce a general high-level switch architecture model that is typically found in automotive switches. Based on this simple model, we highlight the key critical components that may introduce dependencies that might not be obvious on first sight.

2.3.1 General Switch Architecture

A high-level overview of the general architecture of typical Ethernet switches is presented in Figure 2.10 (cf. [135], [33], [34], [121]). Usually, a switch comprises four functional blocks: (1) an ingress stage, (2) a switching stage, which is often divided into a forwarding table, frame handling logic and a switch fabric, (3) an egress stage, and (4) management functionality.

2.3.2 Ingress Stage

One of the main tasks of the ingress stage is to perform integrity checking by evaluating the FCS of each received frame. Frames not passing this check are dropped in this stage.

The ingress stage is also an important component to realize certain QoS mechanisms, such as fault or attack containment, e.g. by implementing some form of ingress rate limiting or Ethernet TSN's ingress filtering and policing mechanisms according to IEEE 802.1Qci (see Section 2.2.3.6 and [81]) to slow down attacks or filter derouted or rogue traffic. Frames not passing filtering and policing are dropped. As IEEE 802.1Qci allows per-stream filtering and

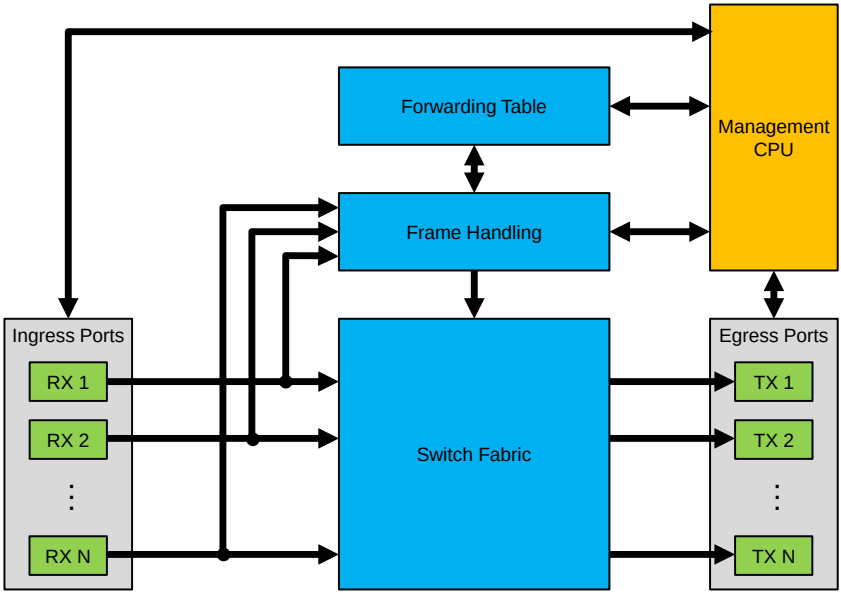


Figure 2.10: High-level switch architecture

policing, frame dropping only affects the violating traffic stream, while keeping others untouched.

Frame dropping negatively affects frame delivery properties and timing behavior. As frame dropping in Ethernet is not reported to the involved communication partners, higher-layer mechanisms to ensure reliable frame delivery are required, e.g. by retransmitting dropped frames (cf. [164]). Retransmissions, however, increase the number of frames being transmitted across the network, potentially resulting in increased interference.

Some switches implement proprietary input filtering and policing, e.g. as a protection mechanism against multicast or broadcast storms. If this is done per traffic class then freedom from interference is violated, as a single traffic stream could affect all other streams in its class. This issue is even worse when this filtering and policing is done on a per-port basis.

2.3.3 Frame Handling

The frame handling block parses the headers of all frames that passed the ingress stage. It extracts the information required to determine via which output port(s) a frame should be transmitted. This is typically its destination MAC address and often its VLAN ID. Which are then used to retrieve the frame's designated output port(s) from the forwarding table. From this information, frame handling generates control signals for the switch fabric.

2.3.4 Forwarding Table

The main feature of switches (in a connected and cycle-free topology) is that they forward frames systematically towards their destination. To do this, they must know via which port each destination can be reached. This information is learnt from the source MAC addresses of incoming frames and is stored in the forwarding table.

The forwarding table is typically implemented as a set associative array. An associative array associates keys with their corresponding values. In the context of Ethernet frame forwarding typically the destination addresses, often combined with the VLAN IDs, are used as keys. The values contain the actual forwarding information, i.e. to which port(s) a frame, whose key is associated with this value, should be forwarded to.

Due to the large key space (2^{48} if only destination addresses are used and 2^{60} if, additionally, VLAN IDs are used), switches cannot provide storage to hold values for all possible keys/value pairs. Instead, switches typically only offer between 256 and 4096 memory locations to hold values, e.g. [31]. Consequently, switches usually use a hash table or (less commonly) a content-addressable memory (CAM) [139] with a RAM to implement their forwarding table [135].

As hash tables use non-injective hash functions, hash collisions may occur and must be handled. In case of a hash collision, a switch typically floods the frame that caused the collision to all ports as an attempt to deliver the frame nonetheless (even though inefficiently). In the timing context, flooding can lead to unexpected traffic on certain output ports, which can have a negative impact on the timing behavior of regular traffic, due to increased interference. For automotive Ethernet switches, the OPEN Alliance requires the collision-free handling of at least 256 forwarding table entries [138].

The frame handling logic (cf. Figure 2.10) configures the switch fabric based on the forwarding information stored in the forwarding table. There are two ways how the forwarding table can be filled: learning and programming.

Switches learn forwarding information by reading the source MAC address (and VLAN ID) field(s) of all incoming frames. This data and the frame's input port are stored in the forwarding table, which essentially carries the information of which MAC address (in a certain VLAN) can be reached via which port. Later, the frame handling logic uses the destination MAC addresses (and VLAN ID) of incoming Ethernet frames to look up in the forwarding table, to which port each frame should be forwarded to. If the forwarding table does not contain any forwarding information for a destination address (and VLAN ID pair) of an incoming frame, then the frame is flooded to all output ports, with the intention that it will reach its destination eventually. The forwarding information for this destination can then be learnt from its answer (if there is any) as described above. This implies that it can take some time until an address is learnt. Note that forwarding information is only learnt if there is enough space available in the forwarding table, i.e., if the CAM is full or if there is a hash collision, no forwarding information is stored. Multicast addresses cannot be learnt directly, as they are not allowed as source MAC addresses (cf. Section 2.1.4). They can, however, be extracted from IGMP [20] or MLD [144] messages, if automatic configuration is desired.

In order to react to topology and end station changes, it must be possible to overwrite obsolete entries. This is usually implemented via some form of aging mechanism, that discards entries that have not been used for a predefined amount of time.

Alternatively, forwarding table entries can be programmed. This has the advantage that forwarding table entries are instantaneously and permanently available, i.e. they are protected against being aged out. Programming is only feasibly in networks with fixed topology (including end stations), as each topology change potentially requires changes to the forwarding tables. This, however, is typically the case for automotive Ethernet networks [123]. Note that, even with programmed forwarding table entries, the lookup mechanism remains the same. Consequently, MAC address combinations, which would result in a hash collision and exceed the available ways for the hash table, cannot be programmed.

2.3.5 Switch Fabric

The switch fabric is responsible to forward frames from any input port to any output port according to the frame handling logic's control information. In order to avoid additional congestion in the switch fabric, the fabric must be able to forward frames at a sufficiently high throughput.

There are different alternatives to implement a switch fabric and, while being bound to the rules imposed by, e.g., IEEE 802.1Q [76] and, in the automotive context, the OPEN Alliance [138], switch vendors have some degree of freedom to implement their switch fabrics. Some switches use a non-blocking full crossbar architecture, which can concurrently forward frames from any input to any output. In cases where multiple frames are directed at the same output port, buffering is required. This buffering is typically implemented at the inputs in the form of virtual output queueing to eliminate head-of-line blocking, e.g. [125]. One drawback of full crossbars, however, is that they have a high hardware cost, as their area and control structures increase with the square of the number of switch ports. Hence, switches with larger port counts, use, for instance, iterative schemes to match inputs to outputs, such as iSLIP [125].

Alternatively, the switch fabric can be implemented by a shared memory architecture, e.g. [101], [121], [30]. Shared memory architectures are typically built around a dual-ported memory. Incoming frames are stored using the memory's write port and outgoing frame are removed via the read port. The corresponding addresses are generated by the frame handling logic. In order to emulate a non-blocking full crossbar, the memory and its arbitration must be fast enough. Particularly, if the switch has N ports with link speeds of r_{TX}, the memory bandwidth must be at least $2Nr_{TX}$. Consequently, shared memory architectures can only be used to implement switches with lower port counts. The arbitration of the memory's read and write ports is typically done in a time-triggered fashion [101], [121].

Ethernet frames can have variable size (see Section 2.1.4). In order to simplify the data path, the arbitration mechanisms, and memory management (e.g. to avoid fragmentation) in the switch fabric, switches often internally split

frame into smaller fixed-sized cells.[16] The switch memory is then divided into pages which match the fixed cell size. Frames occupy an integer number of pages that meet their storage requirements. Typical page sizes are 128 bytes or 256 bytes. Depending on the frame size, this can lead to large overheads (i.e. wasted memory), especially for small frames. This must be considered during buffer space analysis (see Section 4.3.4).

If the memory is full, frames will be dropped. This can be newly incoming frames or stored frames, e.g. if their DEI flag is set (see Section 2.2.1.1.2). If memory is shared without restrictions, freedom from interference cannot be guaranteed, e.g. a bursty (or rogue) stream of low-priority frames could occupy the entire switch memory, causing incoming high-priority frames to be dropped. As already discussed in Section 2.3.2, frame dropping can have a negative impact on the network's timing behavior.

One solution to mitigate this issue is memory partitioning. This could be implemented per-class, i.e. by reserving a certain amount of pages for a given traffic class. While this prevents interference from other traffic classes, it still does not guarantee isolation between traffic streams of the same traffic class. Alternatively, per-stream memory partitioning is imaginable. However, this might result in complex frame handling logic.

In addition to the throughput requirement, the switch fabric should not introduce too much additional delay. This delay is typically measured from the reception of a frame until it is queued in the output ports, i.e. without considering the delay introduced by potential congestion during transmission selection in the output ports. For modern switches, this delay is typically in the single-digit us range ([32] or (estimated) [156]).[17]

2.3.6 Egress Stage

Conceptionally, the egress stage implements the output port model, i.e. the QoS mechanisms (cf. Section 2.2). However, how the output port model is actually implemented is vendor-specific. Shared memory switches typically keep frames in their memory until they are actually selected for transmission. In this case, the queues in the egress stage often only store pointers to the frames in the shared memory.

IEEE 802.1Q only specifies an upper limit of eight traffic classes (cf. Section 2.2.1.3). Actual switches, however, might only support less traffic classes by implementing less queues per output port [31]. If more priority levels than traffic classes are used, traffic classes must be shared. In these cases IEEE 802.1Q [76] gives a recommended mapping of priority levels to traffic classes. Resource sharing between two different priority levels, however, could violate freedom from interference.

Furthermore, some switch vendors make trade-offs between accuracy and overhead, when implementing QoS mechanisms. In the context of credit-based

[16] This is also done in asynchronous transfer mode (ATM) [95] and Networks on Chip (NoC) [39].

[17] This delay can be reduced by cut-through forwarding, i.e. by forwarding a frame as soon as its destination address has been parsed. However, since no integrity checking can be performed with cut-through forwarding, fault containment is limited, i.e. frames with wrong priorities or VLAN memberships due to bit errors could be forwarded, causing unexpected interference.

traffic shaping, instead of replenishing credit bitwise and frequently, they might replenish multiple bits at a time less often. This might lead to corner cases with a negative timing impact. In the context of WRR, some switches limit the weight that can be assigned to a queue, and hence also limit the ratio of link bandwidth distribution between different traffic classes.

2.3.7 Switch Management

Switches often contain a management processor or provide interfaces for an off-chip management processor, which has access to the switch's functional blocks. This processor is typically used to implement certain protocols. SRP is a prominent example, where the processor checks, tracks, and reserves (switch) resources. Also, time synchronization according to IEEE 802.1AS [69] is often implemented with the help of a processor. Often this processor is (additionally) connected to the switch via an Ethernet interface in order to be reachable over the network.

2.4 Higher-Layer Protocols

Ethernet is a layer 2 protocol, i.e. it operates on the data link layer, and delivers frames based on (layer 2) MAC addresses. Technically, Ethernet's addressing and frame delivery mechanisms would be sufficient to implement in-vehicle communication networks. However, network applications rarely access the data link layer directly. Instead, they typically use services provided by higher layers, such as the network layer (layer 3) and the transport layer (layer 4).

The main reason for this is the separation of concerns, i.e. each layer is responsible for a distinct task. While layer 2 protocols can only establish communications within a concrete layer 2 network (such as a local Ethernet network or a VLAN), higher-layer protocols are used to allow the inter-network communication between different layer 2 networks, e.g. inter-VLAN communication or the communication between Ethernet and WiFi (IEEE 802.11). Also, higher-layer protocols typically abstract from layer 2 limitations such as the maximum frame size and add support for application-independent fragmentation (and reassembly) to distribute large data packets over multiple layer 2 frames in a coordinated manner. Another feature that is often implemented by higher-layer protocols is reliable data transport. This is important, as the data link layer often only offers connectionless frame forwarding, i.e. treats each frame independently in a "fire-and-forget" manner. This is the case for Ethernet. In Ethernet frames can also be dropped for various reasons (e.g. due to checksum mismatches or buffer overflows).

Another reason against layer 2 only networking is the separation for performance and security. In large layer 2 networks broadcast traffic can contribute significantly to network congestion. This can be mitigated by separating a network into distinct layer 2 broadcast domains, e.g. by using higher-layer network devices such as routers. Other higher-layer network devices, such as firewalls, can be used to limit or allow access more selectively.

In order to coordinate the information exchange on higher layers, protocols are required. These higher-layer protocols, however, introduce a certain overhead. Each protocol requires additional information to provide its service, e.g. layer 3 addresses or connection management information for reliable data transport. Typically, this information is added as a header to the data to be transmitted. Each additional header implies that more bits must be processed and transmitted, which leads to longer frame processing and transmission times and, ultimately, to longer communication latencies. Some protocols even require a connection setup process and continuous connection management (e.g. by exchanging congestion and flow control information), which can also have a significant timing impact. Despite their overhead, higher-layer protocls (e.g. IP, TCP, and UDP) are typically used for automotive communication. Beside the technical advantages outlined above, flexibility, interoperability (with existing equipment and software), and reuse (of existing and mature gateway technologies, communication stacks, and (debug) tools) are the key factors for this decision.

In the following, we briefly present the most common higher-layer protocols, which are used on top of Ethernet in the automotive domain. We pay special attention to their protocol overheads, as these have an impact on the worst-case timing behavior.

2.4.1 Internet Protocol

The internet protocols IPv4 and IPv6 are connectionless packet-oriented layer 3 protocols.[18] In this section, we discuss the basic principles of IP using IPv4 as an example. Then, we present IPv6 and discuss its main differentiating factors compared to IPv4.

2.4.1.1 Internet Protocol Version 4 (IPv4)

IPv4, [89] provides three basic features: (1) layer 2 independent addressing and inter-network routing, (2) multicast, and (3) fragmentation to abstract from any layer 2 frame size limits.

2.4.1.1.1 Addressing and Routing

As a layer 2 independent protocol, IPv4 defines its own addressing scheme in the form of IPv4 addresses. IPv4 supports 2^{32} unique addresses. If IPv4 is used on top of Ethernet, IPv4 addresses must be mapped to Ethernet MAC addresses to enable communication. This is done by the address resolution protocol (ARP).

From the perspective of a traffic stream, routing is defined to be the process of selecting a path from a source to a destination across (multiple) layer 2 networks. Routing is performed by routers, which interconnect different layer 2 networks. The actual routing decision is typically done based on a routing table,

[18] Note that there is no IPv5. Version 5 has been assigned to the internet stream protocol (ST2+) [43], which never made it past the experimental stage.

which contains information on how to reach various destination networks. Compared to layer 2 forwarding, routing is typically done in software and can involve complex decisions (including the interaction with other routers), while forwarding is usually realized entirely in hardware. This allows routing algorithms to find (optimal) paths in meshed networks (including routing loop prevention), whereas forwarding requires a cycle-free active network topology.

Routing is required to connect an in-vehicle network to other networks. Examples of routing include inter-VLAN routing (between different vehicle domains) or routing between the vehicle network and the Internet (e.g. for over-the-air software updates).

2.4.1.1.2 Multicast

As in Ethernet, multicast can help to better use network resources and ease communication whenever one-to-many communication is required. IPv4 uses special multicast addresses to define multicast groups and supports up to 2^{28} multicast groups, where each group is identified by its address. In order for an IPv4 end station to receive IP multicast traffic from subnets other than its own, it must register this request at a router in its IP subnet via the internet group management protocol (IGMP) [21] or the router must be preconfigured to listen for and route certain multicast addresses.

In Ethernet IPv4 multicast addresses are mapped to special Ethernet multicast addresses. For IPv4 multicast, Ethernet reserves 2^{23} multicast addresses. Hence, each Ethernet multicast address is shared among 32 IPv4 multicast groups.

2.4.1.1.3 Fragmentation

Fragmentation is required whenever IPv4 must transport higher-layer packets, that exceed the maximum payload size of the underlying layer 2 network. While fragmentation allows to distribute large higher-layer packets over (multiple) smaller layer 2 frames, it can have a negative impact on the higher-layer packet loss rate. This is because if any layer 2 frame from a fragmented IPv4 packet is missing, the entire packet will be dropped in the IPv4 stack. The IETF considers IP fragmentation to be fragile [142].

2.4.1.1.4 Overhead

The IPv4 header includes the source and destination IPv4 addresses as well as other fields to support routing and fragmentation. The header is protected by a checksum. Optionally, multiple IPv4 options can be attached to the header. The IPv4 header is at least (i.e. without any options) 20 bytes long [89]. This leads to larger layer 2 frames and hence to longer transmission times and longer interference with other traffic streams. Also, processing the header in layer 3 devices (e.g. routers and end-stations) takes a certain amount of time.

Fragmentation splits a layer 3 packet into multiple layer 2 frames. This must be considered during timing verification, e.g. when deriving end-to-end latency bounds, the time it takes to transport multiple associated layer 2 frames must be computed (cf. Section 3.3.2.2.1). As discussed, fragmentation

can lead to increased packet loss from a higher-layer perspective, as a packet is dropped as soon as one of its fragments is missing or corrupt. This can be mitigated by using retransmission mechanisms, which, however, also have a negative impact on the timing behavior (e.g. [16]).

Before an IPv4 packet (wrapped into an Ethernet frame) can be sent, the MAC address of the destination IP address must be known. IPv4 relies on ARP to dynamically learn the mapping between (unicast) layer 2 and layer 3 addresses, e.g. between Ethernet MAC addresses and IPv4 addresses, so that in the worst-case a preceding exchange of ARP request and ARP response messages must be considered.[19] In (mainly) static automotive networks, this could be mitigated by preprogramming an already populated ARP table into the end stations.[20]

Routing, in contrast to layer 2 forwarding, is typically implemented in software and might not be able to be performed at wire speed. Additionally, the (Ethernet) link(s) to the router can be become a bottleneck as well, especially, if multiple VLANs, which appear independent in a high-level model, are using the same link. Consequently, a proper router model (which can be derived with the components we introduce in Chapter 3) must be included in a formal performance analysis. Note, however, that inside the same IP subnet, routing is not required and the end-to-end latency in the network is determined by layer 2 forwarding.

As discussed, Ethernet offers less multicast group addresses than IPv4 does. Consequently, an Ethernet multicast tree might be the result of a combination of different IPv4 multicast groups, all of which map to the same Ethernet multicast address. This can lead to unanticipated load in network segments which are part of the Ethernet multicast tree but are not part of a given IPv4 multicast tree. Additionally, this imposes a security risk, as traffic might be leaked to untrusted destinations. Introducing cross-layer design rules to constrain multicast group address usage can be used to mitigate this problem.

2.4.1.2 Internet Protocol Version 6 (IPv6)

The IPv4 address space exhaustion was one of the key drivers, which lead to the development of its successor, IPv6, [42]. Other distinctive features of IPv6 compared to IPv4 are: stateless address autoconfiguration (DHCPv6 is also (and complementingly) possible) and duplicate address detection as well as neighbor unreachability detection, a simpler header to make routing more efficient, and efficiency-optimized multicast (i.e. a multicast address encodes whether traffic should stay, for example, within a local site or organization or should be global).

From the point of our discussion, IPv6 provides the same basic functionality and timing properties as IPv4, i.e. layer 3 inter-networking. Compared to IPv4, the IPv6 header [42] has been streamlined for performance, i.e. it only includes the most basic information that almost all packets require. Any additional features can be enabled by using so-called extension headers, which can be

[19] For IPv6 the same argumentation can be applied to NDP.

[20] A preprogrammed ARP table could also improve security, e.g. to impede ARP snooping attacks.

chained to the IPv6 header. Examples of such features include fragmentation support as well as support authentication, data integrity, and data confidentiality [42]. Interestingly, IPv6 dropped support for layer 3 checksums, assuming that other layers (both higher and lower) provide sufficient error detection. A a large fraction of the header is allocated to the IPv6 addresses. Altogether, the IPv6 header adds an overhead of (at least) 40 bytes (without any extension headers).

2.4.2 Transmission Control Protocol (TCP)

TCP, [90] is a transport layer (layer 4) protocol that adds reliable, bidirectional, and connection-oriented communication on top of IP. While IPv4 and IPv6 interconnect end stations, TCP interconnects different applications or processes running on these end stations. Individual applications are distinguished by port numbers, which allows the multiplexing of independent TCP connections over IP. TCP provides four main features: (1) connection orientation, (2) streaming, (3) reliable data transport, and (4) flow control and congestion avoidance.

2.4.2.1 Connection-oriented Communication

A TCP connection between the processes of two end stations must be established via a handshake mechanism. The handshake comprises the exchange of three messages and allows both communication partners to synchronize their state regarding this connection. This state is kept synchronized throughout the connection and is the foundation that enables TCP's other features such as streaming, reliable data transport, and flow control. The exchanged state information includes, for example, sequence numbers and the amount of available buffer space at the receiver.

2.4.2.2 Streaming

TCP provides a streaming interface, i.e. processes hand over their (continuous stream of) data to the communication stack and TCP manages the actual transmission. A process does not need to (and cannot, unless Nagle's algorithm [96] is applied) decide when data is actually transmitted. Internally, TCP buffers the data stream and divides it into segments, adds a header, and passes the segments to IP where they become the payload of IP packets. This segmentation, i.e. how much data is sent and when it is sent, is performed automatically at runtime via heuristics which take into account data availability, flow control, and network congestion information.

2.4.2.3 Reliable Data Transport

Generally, on their way through a network, TCP segments can be lost, duplicated, or arrive in the wrong order. Reliable data transport in TCP is ensured via different measures. Each segment is checked for integrity via a checksum upon arrival and corrupted segments are dropped. Furthermore, TCP acknowledges the reception of segments and each segment includes a sequence number.

The sender detects lost packets via the absence of acknowledgments, i.e. if an acknowledgment is still outstanding after a given amount of time (timeout), and issues a retransmission of the lost segment. Duplicated segments are detected at the receiver through their sequence numbers. Sequence numbers are also used to reorder segments inside the communication stack (if required) for in-order delivery.

2.4.2.4 Flow Control and Congestion Avoidance

Flow control dynamically adjust the data rate of the sender to avoid buffer overflows at the receiver. In TCP, flow control is implemented as a set of sliding windows, one at the sender (send window) and one at the receiver (receive window). The send window size corresponds to the amount of data the sender is allowed to send without acknowledgment from the receiver. Likewise, the receive window size corresponds to the maximum number of frames the receiver can buffer without sending acknowledgments to the sender. With each acknowledgment the receiver informs the sender about how much data it has processed from its receive window. This allows the sender to advance its send window accordingly, so that it never sends more data than the receiver can buffer.

While flow control avoids data loss when the receiver is the bottleneck, i.e. processes data slower than the sender generates it, congestion avoidance avoids data loss in cases when the network is the bottleneck. It dynamically adjusts the data rate of the sender according to an estimation of the current network throughput. In TCP congestion avoidance is implemented by a (sliding) congestion window at the sender, whose size corresponds to the maximum number of unacknowledged segments. A heuristic called slow start is used to dynamically increase the congestion window size until packet loss occurs, in which case the window size is reduced, or until the data rate is dominated by flow control [163], [10].

2.4.2.5 Overhead

The TCP header contains port fields to allow multiplexing connections of multiple applications, a checksum, and fields to support flow control and to keep track of segmentation [90]. If used without options, it adds an overhead of 20 bytes. Similar to IPv4 and IPv6, TCP's header additionally increases the frame length and, hence, their transmission and interference times.

Due to its design, it is hard to give (tight) timing guarantees for TCP. First of all, to exchange data, a connection must be established via TCP's handshake mechanism. This takes 1.5 times the round trip time and must be repeated whenever the connection is lost. Under worst-case conditions, these connection setup times must be considered during communication delay computation. Second, segmentation in TCP is done dynamically at runtime and depends on heuristics, impeding formal analysis. Third, flow control and congestion avoidance act as dynamic traffic shapers and, hence, also have a negative and hard to predict impact on TPC's timing behavior.

2.4.3 User Datagram Protocol (UDP)

UDP, [141] is a connection-less transport layer (layer 4) protocol on top of IP. Like TCP (cf. Section 2.4.2), UDP allows different applications or processes to exchange data. Again, ports are used to distinguish processes. In contrast to TCP, however, UDP's main goal is simplicity, as often not all of TCP's features are needed. It offers two main features: (1) connection-less and (2) datagram-oriented data transport.

2.4.3.1 Connection-less and Datagram-oriented Communication

Its connection-less nature allows UDP to send data without prior connection setup in a "fire-and-forget" manner. However, as UDP does not keep any connection state (there is, for example, no feedback from the receiver), no delivery guarantees can be given. As a consequence, data loss and duplication are not detected and data might arrive at the receiver in a different order than it has been sent. One advantage of connection-less communication, and hence of UDP over TCP, is that multicast and broadcast transmissions can be implemented easily.

UDP's transmission entities are called datagrams. To UDP, datagrams are independent of each other and no assumptions about their relation to each other are made. A sender has full control over which data is packed into a datagram and when datagrams are handed over to the communication stack for transmission. In the automotive context, this is an important property, as it allows critical data, for instance, to be sent immediately.

2.4.3.2 Overhead

UDP datagrams contain port fields to allow multiplexing connections of multiple applications and include a checksum field to provide basic error checking [141]. While the checksum field is always present in the UDP header, its use is optional (in IPv4) and there is no built-in mechanism to handle erroneous datagrams. Instead, UDP relies on higher-layer protocols to handle errors (if required). The UDP datagram header adds 8 bytes of protocol overhead.

2.4.4 Additional Considerations

So far we have focused on the pure protocol overhead. Higher-layer protocols, however, are typically implemented via software components. These software components compete with other software (e.g. applications) for CPU time. Additionally, scheduling constraints have to be taken into account, as often components of the communication stack are executed as part of a periodic task, which introduces a certain reaction time.

2.5 Summary

In this chapter, we introduced Ethernet in the automotive context and provided an overview of the entire communication stack ranging from the relevant physi-

cal layer standards up to higher-layer communication protocols. We highlighted the timing impact of the individual components on the communication path, especially switches, QoS mechanisms, and higher-layer protocols. While the timing impact of switches can often be controlled by careful configuration, Ethernet QoS mechanisms and higher-layer protocols are the main contributors to the overall communication latency.

In this thesis, we focus on the formal timing analysis of QoS mechanisms in Ethernet. In the next chapters we will recapitulate the state of the art of compositional performance analysis, which will be used as the underlying analysis framework for the formal timing analysis of Ethernet TSN standards.

Chapter 3

Compositional Performance Analysis

The main goal of this thesis is to derive upper bounds on the worst-case timing behavior of messages sent in an Ethernet network, e.g. to check whether these messages meet their timing requirements (such as given deadlines).

There are two fundamentally different approaches to evaluate the timing behavior of a system: simulation and formal methods. Simulation is used to observe a system's response to a given set of provided stimuli. It provides valuable insights into the system's behavior, such as the possibility to learn what a system is doing at any point in time or to derive histograms of observable system properties such as the path latency (e.g. [97], [8], [127], [9] in the Ethernet context). However, simulation often requires long simulation runs and typically does not expose all (timing) corner cases. It rather provides an observed worst-case. This renders simulation unsuitable for the timing verification of timing- and safety-critical systems under worst-case conditions. Consequently, in this thesis, we focus on formal performance analysis methods.

3.1 Related Work

In the context of formal performance analysis frameworks, network calculus [116] is one of the prominent frameworks. Network calculus is a system theory for computer networks based on min-plus algebra. It abstracts data arrival by accumulative arrival curves and the service provided by resources as service curves, which give bounds on the data arrival and the behavior of a resource, respectively. Performance bounds (e.g. output arrival curves, end-to-end latency, and backlog) can be derived by evaluating the relations between arrival and service curves through min-plus algebraic operations or direct comparison between the curves. One key property of network calculus (as a result of the underlying min-plus algebra) is the concatenation of a tandem of resources, each with its own individual service curve, into an abstract resource with a new service curve. This new service curve is computed by the convolution of

the individual services curves and can be used to derive better performance bounds, a property known as pay bursts only once.

One challenge with networks (in particular) is the aggregation of traffic streams, i.e. when multiple traffic streams use the same resource. Now, each individual stream (of such an aggregate) typically only receives a fraction of the service (left-over service) offered by the resource (depending on the scheduling strategy). Multiple approaches for the (efficient) analysis of individual traffic streams have been proposed. They mainly differ in how the left-over service that is offered to a traffic stream is derived and how it is used during the analysis of performance bounds. There is no strict order on the tightness or analysis complexity of these approaches.

Total flow analysis [154] only considers aggregate traffic streams. It derives per-resource performance bounds by aggregating the traffic streams that traverse a resource. The end-to-end latency is then bounded by the sum of the per-resource delay bounds on the traffic stream's path. Total flow analysis has been extended in [128] to exploit the fact that the link capacity in communication networks is fixed. This can be used to compute tighter arrival curves for aggregate traffic streams and to compute tighter output arrival curves (by exploiting that the link capacity also implies a certain bound on the maximum service) for individual streams. It is demonstrated in [128] that this improvement yields high accuracy and, at the same time, low analysis complexity (i.e. runtime).

Another way to derive per-stream performance bounds in the presence of stream aggregation is the separated flow analysis [154], which computes per-stream left-over service curves on each resource traversed by a stream and then uses network calculus's concatenation property to derive the stream's concatenated end-to-end service curve, which can, in turn, be used to derive performance bounds.

The pay multiplexing only once analysis approach [155] exploits knowledge of shared path segments when deriving the left-over service for a traffic stream. It first uses network calculus' concatenation property to abstract tandems of resources and then derives the per-stream left-over service on the abstracted resources. This way the multiplexing of interfering streams (that share a certain path segment with the stream under analysis) is only accounted (paid) once.

Handling cyclic dependencies in formal performance analysis frameworks requires special attention. There are two essential approaches to handle (non-functional) cyclic dependencies: (1) handling them within the analysis framework (e.g. by a fixed point iteration) and (2) breaking them (e.g. by ensuring that the impact of cyclically-interfering traffic streams is bounded to a known pattern). The latter can be achieved by introducing regulators [19], [115] into the communication paths of traffic streams that reshape their arrival curves. The work in [175] discusses handling of cyclic dependencies in network calculus and presents an improved total flow analysis that handles both systems with and without cyclic dependencies. It, particularly, focuses extensively on the effect of regulators (per-flow and interleaved) and their deployment strategies (partial (i.e. at strategic locations) and total (i.e. everywhere)) to break cyclic dependencies versus fixed point iterations. To this end, it also introduces an

algorithm for partial regulator placement. The discussion in [175] is underpinned by the evaluation of synthetic and industrial networks that show under which conditions the different deployment strategies of regulators are most beneficial.

Real-time calculus [173], [52], [99] is a popular performance analysis framework, which is based on network calculus' principles, focusing on the formal performance analysis of distributed real-time systems.

Compositional performance analysis (CPA) [66] is another popular analysis approach. With the goal to reduce analysis complexity, CPA breaks down a system into individual resources (e.g. processors, communication buses, or Ethernet switch ports), which are analyzed in isolation. The results of these resource-level analyses are then combined via abstract interfaces in an (iterative) system-level analysis, which is also capable to handle cyclic dependencies. In order to remain conservative, certain inter-resource dependencies and correlations have to be overapproximated during the isolated resource-level analyses. As a consequence, the obtained analysis results are also typically an overapproximation of the actual worst case behavior.

On the resource-level, CPA allows to use analyses from classic real-time scheduling analysis research, e.g. [118]. Event models are used to abstract the communication between resources, by describing how often a task mapped to a resource is activated. It has been introduced by [148] and has, since then, received a substantial number of contributions widening its scope, e.g. [153] (including the introduction of a pay bursts only once path latency analysis), [151], [133], [15], [45].

In this thesis we focus on extending CPA by resource-level analyses for automotive Ethernet transmission selection mechanisms. Since CPA is an essential building block for these analyses, we recapitulate its key modeling concepts, definitions, and proofs in this chapter.

3.2 System Model

The system model provides the modeling foundation for compositional performance analysis. It abstracts a system's hardware and software components as well as the timing behavior of the interactions between these components. We follow the approach taken in [161], [134], and [15] and distinguish between a structural model and a timing model.

3.2.1 Structural Model

The structural model abstracts from the underlying hardware and software architectures by defining a model that allows to reason about the timing behavior of a system. Particularly, it defines the system's topology, how much service is required by individual service consumers, and which service consumers concur for service.

Definition 2 (Platform)**.** *A platform is a set of interconnected resources.*

The interconnection between resources is defined by the platform architecture (e.g. which ECU is connected to which bus or the network topology).

Resources abstract the actual processing or communication components (CPUs, buses, switch ports) of a platform.

Definition 3 (Resource)**.** *A resource provides service according to a scheduling policy.*

A resource can only serve one task at a time and, hence, is a point of contention. The scheduling policy, implemented by a scheduler, defines how the resource's service is distributed among tasks, e.g. during times of contention.

Definition 4 (Scheduler)**.** *A scheduler defines how the service of a resource is partitioned.*

A task is the smallest (abstract) entity of computation or communication that realizes a certain function, e.g. processing of sensor data or transmission of a message. Tasks are mapped (assigned) to resources.

Definition 5 (Task)**.** *A task executes on a resource and consumes service.*

Tasks may have additional scheduling parameters, e.g. a priority, which is typically used specify how they shall be treated by a scheduler.

An ensemble of cooperating tasks forms an application.

Definition 6 (Application)**.** *An application is a set of interconnected tasks, forming a task graph exhibiting the inter-task communication dependencies.*

We say that a task i depends on another task j, if i requires data from j in order to execute. Inter-task communication dependencies are abstracted by directed edges in the task graph. The input and output degrees of tasks are either 0 or 1. To model complex applications with diverging and rejoining task chains, special fork and join structures can be used. A platform may contain multiple (concurrent) applications.

3.2.2 Timing Model

The timing model abstracts the timing behavior of a system. We start by abstracting the timing behavior of a task. In context of CPA, events are used to abstract the activation of a task [153].

Definition 7 (Event)**.** *An event is an occurrence that has an importance for a system.*

Events are processed in the order they arrive. In actual systems, events typically are the arrival of data (e.g. an Ethernet frame). If a task is activated, it consumes the activating event and can execute. An activated task may not be able to execute immediately, e.g. due to congestion. In this case, the execution is deferred, i.e. task activations are never lost.

The amount of service a task consumes per execution is determined by its execution time, i.e. the time required by the task to complete its execution if it is executed on a given resource without any interference. This implies that

a task's execution time typically depends on the (speed of the) resource the task is executing on. Furthermore, the execution time typically depends on its concrete input data and on its internal state as well as the state of the resource (e.g. cache, pipeline, branch prediction) [179]. In a formal analysis, we abstract any execution time variance by only using lower and upper bounds.

Definition 8 (Best-Case and Worst-Case Execution Times)**.** *The best-case and worst-case execution times C_i^- and C_i^+ of a task i are a lower and upper bound of its execution time.*

Once a task completes its execution, it produces an event, which, in turn, could activate another task. The communication dependencies (cf. Definition 6) between two tasks are typically defined via the production and consumption of events (cf. Definition 21), i.e. if the events from the completion of a task i activate a task j, we say that task j is a dependent task of task i. Between its activation and completion, a task neither consumes nor produces events.

A task's timing behavior is typically defined via a sequence of events and their timing relation to each other.

Definition 9 (Event Stream)**.** *An event stream is a sequence of semantically related events.*

Examples of event streams are all events produced by a (timer) interrupt or all events that activate a certain task.

As it is typically impossible to evaluate a possible event occurrences in an event stream to determine the timing behavior of a task, event models are used to abstract the timing behavior of the events in an event stream to its corner cases.

3.2.2.1 Event models

Event models abstract the timing behavior of event streams by defining lower and upper bounds on the number of events arrivals in a certain time interval or time distance between events. All event streams, whose event arrivals (or distances) stay within these bounds satisfy the event model and can, hence, be abstracted by the event model. Together with the execution time bounds from Definition 8, this allows a data-independent analysis of a system's timing corner cases.

Event models can be defined in the continuous time interval domain by a set of arrival functions [66], [148].

Definition 10 (Arrival Functions)**.** *The upper and lower arrival functions $\eta_i^+(\Delta t)$ and $\eta_i^-(\Delta t)$ of a task i describe an upper and lower bound on the accumulated number of events in any right half-open time interval $[t, t + \Delta t)$.*

In some cases, we also need a bound on the maximum number of events in any closed time interval $[t, t + \Delta t]$. We indicate this by using the notation $\eta_i^{+]}(\Delta t)$.

Alternatively, an event model can be described in the discrete event domain by a set of distance functions [66], [148].

Definition 11 (Distance Functions)**.** *The minimum and maximum distance functions $\delta_i^-(q)$ and $\delta_i^+(q)$ of a task i describe the minimum and maximum time interval between the first and the last event in any sequence of q consecutive events.*

Arrival functions and distance functions are pseudo-invers. Hence, if one set of functions is known, we can derive the other set [46], [15]. To derive arrival functions from distance functions we use [15]:

$$\eta_i^+(\Delta t) = \begin{cases} 0 & \text{if } \Delta t = 0 \\ \max\limits_{q \in \mathbb{N}^+} \left\{ q \,|\, \delta_i^-(q) < \Delta t \right\} & \text{otherwise} \end{cases} \tag{3.1}$$

$$\eta_i^-(\Delta t) = \min_{q \in \mathbb{N}^+} \left\{ q \,|\, \delta_i^+(q+2) > \Delta t \right\} \tag{3.2}$$

and vice-versa:

$$\delta_i^-(q) = \inf_{\Delta t \geq 0 \wedge \Delta t \in \mathbb{R}} \left\{ \Delta t \,|\, \eta_i^+(\Delta t) \geq q \right\} \tag{3.3}$$

$$\delta_i^+(q) = \sup_{\Delta t \geq 0 \wedge \Delta t \in \mathbb{R}} \left\{ \Delta t \,|\, \eta_i^-(\Delta t) < q \right\} \tag{3.4}$$

Sometimes one notation leads to more intuitive or compact formulations than the other. Hence, in the following, we use both notations interchangeably.

The upper event arrival functions $\eta_i^+(\Delta t)$ are subadditive [116], i.e.:

$$\forall \Delta t_1, \Delta t_2 : \eta_i^+(\Delta t_1 + \Delta t_2) \leq \eta_i^+(\Delta t_1) + \eta_i^+(\Delta t_2) \tag{3.5}$$

Conversely, the minimum distance functions $\delta_i^-(q)$ are superadditive, if shifted by 1 [134], i.e.:

$$\forall q_1, q_2 : \delta_i^-(q_1 + q_2 + 1) \geq \delta_i^-(q_1 + 1) + \delta_i^-(q_2 + 1) \tag{3.6}$$

Both upper event arrival functions and minimum distance functions are monotonically increasing, since the number of event arrivals in any time interval remains constant or grows if this time interval is widened and, vice versa, the minimum distance between any sequence of events remains constant or grows when a larger sequence of events is considered.

Often, it is inefficient or impossible to define arbitrary event models throughout their entire domain and, instead, they are or can be defined only up to a certain $\widehat{\Delta t}$ or $\hat{q}$. In these cases, the sub- and superadditive properties of upper event arrival and minimum distance functions can be exploited to conservatively extrapolate $\eta_i^+(\Delta t)$ and $\delta_i^-(q)$ beyond $\widehat{\Delta t}$ and $\hat{q}$ based on previous values [116], [153].

3.2.2.2 Parameterized Event Models

Many event sources emit events following a fixed pattern that can be modeled by a set of parameters. In the automotive domain or control systems, for instance, communication is often periodic. In these cases, event emittance can

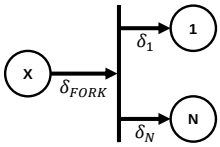


Figure 3.1: Fork structure

often be characterized by a base period, a given uncertainty (jitter) of event arrival relative to the period, and a minimum distance between consecutive events in cases where the jitter is larger than the period [148], [45].

Definition 12 (PJd Event Model)**.** *The number of events of a task i that are generated periodically with a period P_i, a period jitter J_i, and a minimum distance $d_i^{min} \leq P_i$ between any two consecutive events, can be bounded by the following distance functions:*

$$\delta_i^-(q) = \begin{cases} 0 & \text{if } 0 \leq q < 2 \\ \max\left\{(q-1)P_i - J_i, (q-1)d_i^{min}\right\} & \text{otherwise} \end{cases} \tag{3.7}$$

$$\delta_i^+(q) = \begin{cases} 0 & \text{if } 0 \leq q < 2 \\ (q-1)P_i + J_i & \text{otherwise} \end{cases} \tag{3.8}$$

3.2.2.3 Multiple Outputs

Currently, our application model does not allow tasks with multiple outputs. In actual systems, however, this is commonly required. Multicast or broadcast communication (in Ethernet), for example, require a task to distribute its output data to multiple successors. In order to handle multiple outputs, we extend our system model by a fork structure (see Figure 3.1). A fork has a single input and distributes all of its input events to multiple outputs.

The fork semantic used in this thesis is defined as follows (cf. [153]):

Definition 13 (Fork)**.** *A fork distributes its input event model unmodified to all of its outputs. The output event model of each output of a fork is bounded by the following distance functions:*

$$\forall i \in succ(FORK) : \delta_i^-(q) = \delta_{FORK}^-(q) \tag{3.9}$$

$$\forall i \in succ(FORK) : \delta_i^+(q) = \delta_{FORK}^+(q) \tag{3.10}$$

where $\delta_{FORK}^-(q)$ is the input event model of the fork and $succ(FORK)$ is the set of the fork's successor tasks.

3.3 Analysis

So far we defined a system model comprising a structural model and a timing model, i.e. how the resources in a system are interconnected, how tasks are

mapped to these resources, and how they interact with each other. In this section, we summarize how the information from this system model can be used to derive bounds on the corner-case timing behavior of a system.

On the highest level, the analysis is split into two iterative subanalyses: the resource-level analysis and the system-level analysis. The resource-level analysis considers each resource in isolation and derives resource-level timing bounds. The impact of the resource-level timing behavior on other system components is abstracted by event models. The system-level analysis combines the resource-level timing behavior of different resources by propagating their event models according to their communication dependencies in order to derive bounds on the system-level timing behavior.

3.3.1 Resource-Level Analysis

The resource-level analysis considers resources in isolation and derives for each task individually bounds on its timing behavior such as the worst-case response time, activation backlog, or output event models. The analysis relies on two functions: (1) A function determining how many consecutive events (and their corresponding task activations) can influence each other from a timing perspective. This function is used as a stopping condition for the analysis. (2) A function to actually derive the timing behavior of individual events (and their corresponding task activations).

In [118] (and later [177]), the longest level-i busy period, i.e. the longest time interval during which a resource is busy processing events of a task with priority i (including all interference), is introduced to reason about how many consecutive events can influence each other. By definition, the longest level-i busy period can be used as a stopping condition, i.e. it is sufficient to only evaluate all events that fall into the longest level-i busy period in order to find the worst-case timing behavior [118]. In order to derive the actual timing behavior of individual events, [118] and [177] introduce the multiple-event busy time[1], which yields the longest time it takes a resource to process a given number of consecutive events.

All [118], [177], and [153] only consider static-priority preemptive (SPP) scheduling and could show that, for SPP, the stopping condition does not need to be computed explicitly, but can instead be derived implicitly while computing the multiple-event busy time. However, this implicit stopping condition does not work for all schedulers such as non-preemptive schedulers (which are typically used for transmission selection in Ethernet, see Section 2.2).[2] This has been demonstrated for the non-preemptively scheduled CAN bus in [41].

Recently, [45] and [15] proposed to redefine the formal approaches introduced by previous resource-level analyses into a unified framework that can be applied to any scheduler. First, [45] introduces a generalized definition of when a resource is busy with respect to a given task that does not rely on the (often

[1] While [118] and [177] introduced this function, the name was coined later by [153].

[2] Note that the explicit stopping condition, i.e. to investigate all events in the longest level-i busy period [118], is correct and can be applied to all schedulers. Only the analysis (shortcut) proposed for SPP by [118], [177], and [153] cannot be applied to all schedulers.

vague or implicit) definition of when a resource is idle and, hence, can also be applied to non-work-conserving schedulers. Second, [45] renamed the longest level-i busy period to scheduling horizon[3] and defined it based on the new busy definition. Finally, [45] introduces the multiple-event processing time, which is a redefinition of the multiple-event busy time from [118], [177], [153] based on the scheduling horizon. With this redefinition, the multiple-event processing time can be applied to all schedulers. A thorough discussion of this framework and a comparison to previous notations can be found in [45].

In this thesis, we use the notation proposed by [45] and [15], which has been summarized and extended in [7]. This notation will be formally introduced in the remainder of this section.

3.3.1.1 Multiple-Event Scheduling Horizon

We start with the scheduling horizon, which is used to reason about whether a number of consecutive events can influence each other from a timing perspective.

Definition 14 (Multiple-Event Scheduling Horizon)**.** *The multiple-event scheduling horizon $S_i(q)$ of q consecutive activations of a task i is the largest right half-open time interval that starts with the arrival of the first event that leads to the first task activation and ends after the completion of the q-th activation just when an additional hypothetical activation of task i could receive infinitesimal service.*

The theoretical construct with the additional activation is used to reason about whether a resource is busy regarding the currently analyzed task i. As the additional activation is hypothetical, it does not need to match the input event model of the currently analyzed task i. The infinitesimal execution time implies that the additional activation can only fit between two activations (of any two tasks) when there are no backlogged activations of tasks that could delay the processing of this hypothetical activation. In priority-based scheduling, for instance, this includes queued activations of task i itself but also queued activations of (interfering) tasks of higher priority. However, if a normal task activation arrives just after the last of all backlogged activations (among all tasks) completes, it would fit in between. This implies that a resource can be busy regarding a task i even if this task currently has no queued activations, e.g. due to interference from other tasks or scheduling artifacts such as waiting for the next TDMA slot (cf. the definition of level-i busy period in [118]). See [45] for a discussion and illustrations.

In summary, the scheduling horizon indicates whether a sequence of q consecutive events can impact the $(q+1)$-th event [45]. Hence, it can be used to explicitly formulate a stopping condition, i.e. an upper bound on the number of task activations that must be considered during a resource-level analysis to find the worst-case timing behavior.

[3] The name has been chosen to better reflect its purpose, i.e. to reason about how many events must be investigated during the analysis.

Definition 15 (Maximum Number of Events in the Scheduling Horizon)**.** *The maximum number $\hat{q}_i$ of events in the scheduling horizon of a task i equals the maximum number of events that arrive during the scheduling horizon of their respective predecessors:*

$$\hat{q}_i = \max_{q>1} \left\{ q \,|\, \delta_i^-(q) \leq S_i(q-1) \right\} \tag{3.11}$$

If $\hat{q}_i$ does not exist, i.e. if the timing of each new event is influenced by its predecessors, task i is not schedulable. Note that the longest level-i busy period corresponds to $S_i(\hat{q}_i)$ (cf. [15]).

3.3.1.2 Multiple-Event Processing Time

Next, we formally define the multiple-event processing time, which is used to derive bounds on the worst-case timing behavior of the individual tasks of the resource under analysis. It provides bounds on the time that is required to process a given amount of consecutive task activations as long as these activations can influence each other, i.e. arrive within the scheduling horizon of their predecessors.

Definition 16 (Multiple-Event Processing Time)**.** *The minimum and maximum multiple-event processing times $B_i^-(q)$ and $B_i^+(q)$ of a task i are the lower and upper bounds on the time interval between the arrival of the first and the completion of the q-th task activation under the assumption that all but the first activating event arrive within the scheduling horizon of their respective predecessors.*

Note that $B_i^+(q)$, as the upper bound, includes all effects that can impact the timing behavior of a task, including backlogged activations of the task itself, interference and blocking from other tasks, as well as scheduler effects such as TDMA delays or context switch overheads. The multiple-event processing time is only defined for events that can influence each other, i.e. for events $q \leq \hat{q}_i$.

While $B_i^+(q)$ is typically scheduler-dependent (see Section 3.3.1.5), a scheduler-independent lower bound for $B_i^-(q)$ is given by qC_i^-, since every activation requires at least its minimum execution time to complete.

3.3.1.3 Critical Instant

The size of the longest multiple-event scheduling horizon or the longest multiple-event processing time depends on the event arrival pattern of the task under analysis, its interferers, and (sometimes) the scheduler state. The event arrival scenario along with the scheduler state that lead to the longest multiple-event scheduling horizon or the longest multiple-event processing time is called the critical instant scenario [120], [118].

Definition 17 (Critical Instant Scenario)**.** *The critical instant scenario of a task i is a scenario in which the event arrivals of task i, the event arrivals of all its interferers, and the scheduler state are aligned such that the processing of task i is maximally delayed.*

The task arrivals for the critical instant scenario are typically derived from each task's input event model. Often a critical instant scenario can be constructed by assuming that all events (and their corresponding task activations) arrive as fast as possible [120], [118]. Depending on the scheduler, however, sometimes multiple event arrival scenarios must be evaluated until the critical instance scenario for a particular task activation q is found (see FIFO scheduling in Section 3.3.1.5.2). Often it is (computationally) easier to provide a conservative upper bound for the multiple-event processing time from an artificial event arrival constellation (e.g. one that may ignore event model constraints) instead of evaluating a multitude of critical instant scenario candidates (cf. Theorem 12 in Section 3.3.1.5.2).

3.3.1.4 Resource-Level Analysis Results

On the resource level, the multiple-event processing time can be used to derive bounds on the timing corner cases of each task and on their memory requirements.

3.3.1.4.1 Timing Corner Cases

The response time of a task is an important metric to evaluate if a task meets its timing constraints.

Definition 18 (Response Time)**.** *The response time of a particular task activation is the time from the arrival of the event leading to the task activation until the task activation is completed.*

We are particularly interested in the worst-case and best-case response times of a task.

Theorem 1 (Worst-Case Response Time of the q-th Task Activation)**.** *The worst-case response time of the q-th activation of a task i is upper bounded by:*

$$R_i(q) = B_i^+(q) - \delta_i^-(q) \tag{3.12}$$

Proof. Follows directly from Definitions 11 and 16. □

Theorem 2 (Worst-Case Response Time)**.** *The worst-case response time (over all activations) of a task i is upper bounded by:*

$$R_i^+ = \max_{1 \le q \le \hat{q}_i} \{R_i(q)\} \tag{3.13}$$

Proof. Follows directly from Theorem 1 and Definition 15. □

Theorem 3 (Best-Case Response Time)**.** *A lower bound of the best-case response time of a task i is its minimum execution time:*

$$R_i^- = C_i^- \tag{3.14}$$

Proof. A task can never execute faster than its minimum execution time. □

The uncertainty between the worst-case and best-care response times is called the response time jitter. It plays an important role during the derivation of the output timing behavior of a task (see Section 3.3.1.4.3).

Definition 19 (Response Time Jitter)**.** *The response time jitter can be upper bounded by the difference between the worst-case and the best-case response times:*

$$J_i^R = R_i^+ - R_i^- \tag{3.15}$$

3.3.1.4.2 Activation Backlog

In addition to providing a sufficient amount of service so that all tasks meet their timing requirements, resources must also provide a sufficient amount of memory to store all task activations that cannot be immediately be served (e.g. in the presence of congestion). This is of particular interest in the context of Ethernet, as here congestion is very likely due to asymmetric (often service-oriented) communication, e.g. one server with multiple clients and memory is scarce due to power and price constraints.

Theorem 4 (Activation Backlog)**.** *The maximum number of concurrently backlogged activations of a task i is bounded by:*

$$b_i^+ = \max_{1 \leq q \leq \hat{q}_i} \left\{ \eta_i^+ \left(B_i^+(q) \right) - q + 1 \right\} \tag{3.16}$$

Proof. It takes $B_i^+(q)$ time units to process q events. During this time an accumulated number of $\eta_i^+(B_i^+(q))$ events arrive. As the q-th event is currently being processed, $q-1$ events must have been processed already and do not need to be buffered anymore. Note that at most $\hat{q}_i$ events must be considered as more events cannot lead to backlogs larger than those that have already been observed for $1 \leq q \leq \hat{q}_i$. □

3.3.1.4.3 Output Event Models

Similar to (input) event models, which abstract the timing behavior of task activations, output event models abstract the timing behavior of task completions, e.g. the distance between the first and the last task completion of any sequence of q task completions. They can be derived for each task from its input event model and its corner-case timing during scheduling. Output event models are important during system-level analysis, as they are the interfaces between dependent tasks, i.e. the output event model of a task becomes the (new) input event model of its dependent task(s) (see Section 3.3.2).

The interference that a task experiences when executing on a resource typically causes its jitter to grow, i.e. in the worst-case the events produced by the task move closer together and in the best-case the best-case events produced by the task move further apart. This effect is captured by the task's output event model (cf. [66] and [45]).

Theorem 5 (Output Event Model Based on Response Time Jitter)**.** *The output event model of a task i with an input event model bounded by the distance functions $\delta^-_{in,i}(q)$ and $\delta^+_{in,i}(q)$ and a response time jitter J^R_i can, for $q \geq 2$, be bounded by:*

$$\delta^-_{out,i}(q) = \max\left\{(q-1)C^-_i\,, \delta^-_{in,i}(q) - J^R_i\right\} \tag{3.17}$$

$$\delta^+_{out,i}(q) = \delta^+_{in,i}(q) + J^R_i \tag{3.18}$$

Proof. We start with Eq. (3.17). Task i requires at least C^-_i time units to execute. If we assume that only task i is executing, the end of its first execution (which generates the first output event) is at least $(q-1)C^-_i$ apart from the end of its q-th execution (which generates the q-th output event). This yields the first term in the maximum of Eq. (3.17). The input event model of task i specifies that in any sequence of q activations of task i there are at least $\delta^-_{in,i}(q)$ time units between the first and the last (q-th) activation. Let us assume that the first of these activations occurs at time t_1. Given the best-case and worst-case response times R^-_i and R^+_i of task i, the minimum time interval between the end of the first and the end of the consecutive q-th activation of task i, which produce the first and q-th output events, can be bounded by $t_1 + \delta^-_{in,i}(q) + R^-_i - t_1 - R^+_i = \delta^-_{in,i}(q) + R^-_i - R^+_i$ and is independent from t_1. Applying Eq. (3.15) yields the second term in the maximum of Eq. (3.17). As both bounds are conservative, we can take the maximum to bound the minimum distance between q output events.

Analogously, the maximum distance between the first and the last event in any sequence of any q consecutive output events of a task i is $t_1 + \delta^+_{in,i}(q) + R^+_i - t_1 - R^-_i = \delta^+_{in,i}(q) + R^+_i - R^-_i$. Applying Eq. (3.15) yields Eq. (3.18). □

Theorem 5 bounds output event models by using an upper bound of the response time jitter (see Definition 19) to bound the new distances for every consecutive number of output events q. In [153] it has been shown that the accuracy of the output event models could potentially be improved by evaluating the distance between any consecutive number of q output events individually based on the multiple-event processing time.

Theorem 6 (Output Event Model Based on Multiple-Event Processing Time)**.** *The output event model of a task i with an input event model bounded by the distance functions $\delta^-_{in,i}(q)$ and $\delta^+_{in,i}(q)$ and a maximum and minimum multiple-event processing times $B^+_i(q)$ and $B^-_i(q)$ can be bounded by:*

$$\begin{aligned}\delta^-_{out,i}(q) = \max\Bigg\{ &\min_{1\leq k\leq \hat{q}_i}\left\{\delta^-_{in,i}(q+k-1) + B^-_i(1) - B^+_i(k)\right\}, \\ &(q-1)C^-_i\Bigg\}\end{aligned} \tag{3.19}$$

$$\delta^+_{out,i}(q) = \max_{1\leq k\leq \hat{q}_i}\left\{\delta^+_{in,i}(q-k+1) + B^+_i(k) - B^-_i(1)\right\} \tag{3.20}$$

Proof. See [153] for a formal proof. The idea behind the theorem is to evaluate the minimum and maximum distance between the first and the q-th task completion for every event k in the scheduling horizon. For Eq. (3.19) the minimum distance between q output events is computed by assuming that the first k events require the longest time to be processed ($B_i^+(k)$), while the last (($q+k$)-th) event requires the shortest ($B_i^-(1)$). As in Eq. (3.17), q consecutive output events cannot be closer together than $(q-1)C_i^-$. Similarly, Eq. (3.20) can be proven by maximizing the distance between the first and the q-th event based on the multiple-event processing time. □

Note that Theorem 6 does not necessarily yield tighter event models than Theorem 5 for all schedulers. One prominent exception is FIFO scheduling. As both Theorem 5 and Theorem 6 are conservative, we can use the tighter one when deriving output event models [1].

$$\tilde{\delta}_{out,i}^-(q) = \max\left\{\delta_{out,i}^-(q), \tilde{\delta}_{out,i}^-(q)\right\} \tag{3.21}$$

$$\tilde{\delta}_{out,i}^+(q) = \min\left\{\delta_{out,i}^+(q), \tilde{\delta}_{out,i}^+(q)\right\} \tag{3.22}$$

3.3.1.5 Local Analyses of Common Schedulers

In this section, we present how the formalism developed so far can be used to implement resource-level analyses for common schedulers. As the focus of this thesis is the formal timing analysis of Ethernet, we limit our presentation to schedulers that will later be used as the building blocks for Ethernet-specific schedulers.

As frame transmission in Ethernet is non-preemptive, these schedulers are non-preemptive as well. For non-preemptive schedulers, the queueing delay plays a key role when deriving the multiple-event processing time.

Definition 20 (Multiple-Event Queueing Delay)**.** *The multiple-event queueing delay $Q_i(q)$ of the q-th activation of a task i is the longest time interval from the arrival of the first event that activates task i until the q-th activation of task i starts executing.*

3.3.1.5.1 Static-Priority Non-Preemptive (SPNP) Scheduling

In static-priority non-preemptive scheduling, each task is assigned a unique priority. If more than one task requests service from a resource, an SPNP scheduler uses the task's priorities to determine which task to execute. Once a task has started executing, it cannot be preempted by any other task until its completion. An SPNP scheduler is often found in Ethernet switches as the final stage of a transmission selection mechanism (cf. Section 2.2)

Theorem 7. *The multiple-event scheduling horizon for the q-th activation of task i under SPNP scheduling is upper bounded by*

$$S_i(q) = qC_i^+ + \max_{j \in lp(i)}\left\{C_j^+\right\} + \sum_{j \in hp(i)} \eta_j^+\left(S_i(q)\right) C_j^+ \tag{3.23}$$

where $lp(i)$ is the set of tasks with lower priorities that that of task i and $hp(i)$ is the set of tasks with higher priorities than that of task i.

Proof. In order to find the multiple-event scheduling horizon, we assume a critical instant scenario where the event arrival sequences (that lead to task activations) of task i and its interfering tasks start simultaneously [120] and we further assume that all interfering tasks arrive as soon as possible, i.e. we bound their arrival by their respective upper arrival functions. Note that the order, in which the interfering tasks arrive, has no impact on the result [100].

The scheduling horizon of the q-th activation of task i ends when an additional hypothetical activation of task i would receive infinitesimal service (cf. Definition 14), i.e when the resource is no longer busy regarding the q-th activation of task i. This is case when all q activations of task i have completed their execution (first term) and all activations of higher-priority tasks (i.e. tasks in $hp(i)$) that arrived within the scheduling horizon of the q-th activation of task i and would prevent the aforementioned additional hypothetical activation of task i from receiving infinitesimal service, i.e. that would keep the resource busy from task i's point of view (cf. [41] and [45]), have been served (third term)[4]. Due to non-preemptiveness, a single lower-priority task might be activated just before task i and its interferers and, hence, contribute to the multiple-event scheduling horizon of task i. In the worst-case, this is the lower-priority task with the longest execution time (second term). Only then an additional hypothetical activation of infinitesimal small execution time could be processed. Thus, the theorem follows (cf. [45]). □

There is no analytical solution to Eq. (3.23), due to the fact that $S_i(q)$ occurs on both sides of Eq. (3.23) and due to the discontinuities introduced by the arrival functions. However, Eq. (3.23) can sill be solved.

Theorem 8. *If a fixed point exists, Eq. (3.23) can be solved by iteration, i.e.*

$$S_i^{(n+1)}(q) = qC_i^+ + \max_{j \in lp(i)} \left\{ C_j^+ \right\} + \sum_{j \in hp(i)} \eta_j^+ \left(S_i^{(n)}(q) \right) C_j^+$$

with $S_i^{(0)}(q) = qC_i^+$ until $S_i^{(n+1)}(q) = S_i^{(n)}(q)$.

A suitable stopping condition must be given, to abort the iteration in case Eq. (3.23) should not converge (within a predefined amount of time or exceeds certain bounds).

Proof. See [161] for a detailed discussion and proof on the convergence conditions and minimality of fixed points in the context of CPA. □

As mentioned before, the multiple-event queueing delay is required to bound the multiple-event processing time under non-preemptive scheduling policies.

[4] As the multiple-event scheduling horizon is defined based on the busyness of a resource, this may even include activations of higher-priority tasks that arrive during the execution of the q-th activation of task i. While in non-preemptive scheduling, these activations cannot delay the q-th activation of task i, they may have a (negative) impact on later activations of task i, as discussed and illustrated in [41], [45], and Section 3.3.1.1.

Theorem 9. *The multiple-event queueing delay for the q-th activation of task i under SPNP scheduling is upper bounded by*

$$Q_i(q) = (q-1)C_i^+ + \max_{j \in lp(i)} \left\{ C_j^+ \right\} + \sum_{j \in hp(i)} \eta_j^{+]} \left(Q_i(q) \right) C_j^+ \tag{3.24}$$

Proof. The proof follows the argumentation of [41]. The critical instant scenario is constructed as in Theorem 7. The q-th activation of task i can be executed when its $q-1$ predecessors have completed their execution (first term) and all activations of its interfering tasks (i.e. tasks in $hp(i)$) that arrived within the queueing delay of the q-th activation of task i have completed execution (third term). As for the scheduling horizon (cf. Theorem 7), we have to assume that, in the worst-case, an activation of the lower priority task with the longest execution time interferes with task i at the beginning of its queueing delay. In the worst-case, a higher-priority task is activated by an event that arrives just at the end of the multiple-event queueing delay. In this case, we conservatively assume that this activation is executed before the q-th activation of task i [22] (recall from Definition 20 that the multiple-event queueing delay covers the time *until* the q-th activation of task i is considered by the scheduler). Consequently, we use $\eta_j^{+]}(Q_i(q))$, i.e. the closed-interval version of the upper arrival function, when bounding the higher-priority interference. □

With the help of Theorem 8, Eq. (3.24) can be solved by iteration.
Now, we can bound the multiple-event processing time.

Theorem 10. *The multiple-event processing time for the q-th activation of task i under SPNP scheduling is upper bounded by*

$$B_i^+(q) = Q_i(q) + C_i^+ \tag{3.25}$$

Proof. Follows directly from Definitions 20 and 16. □

3.3.1.5.2 First In - First Out (FIFO) Scheduling

A FIFO scheduler serves a set of tasks based on the order in which the tasks are activated. As with SPNP, a task that has started executing cannot be preempted by any other task until it has completed its execution. FIFO scheduling is often used to implement the per-class queues in Ethernet switch ports (cf. Section 2.2).

Theorem 11. *The multiple-event scheduling horizon for the q-th activation of task i under FIFO scheduling is upper bounded by*

$$S_i(q) = qC_i^+ + \sum_{j \in fifo(i)} \eta_j^+ \left(S_i(q) \right) C_j^+ \tag{3.26}$$

where $fifo(i)$ is the set of tasks that interfere with task i in a FIFO fashion.

Proof. The critical instant scenario is constructed as in Theorem 7. The scheduling horizon of the q-th activation of task i ends when all q activations of task i have completed their execution (first term) and all activations of its interfering tasks (i.e. tasks in $fifo(i)$) that arrived within the scheduling horizon of the q-th activation of task i have completed execution (second term). Only then an additional hypothetical activation of infinitesimal small execution time could be processed (cf. [15]). □

With the help of Theorem 8, Eq. (3.26) can be solved by iteration.

Theorem 12. *The multiple-event queueing delay for the q-th activation of task i under FIFO scheduling is upper bounded by*

$$Q_i(q) = (q-1)C_i^+ + \sum_{j \in fifo(i)} \eta_j^{+]}\left(Q_i(q)\right) C_j^+ \tag{3.27}$$

Proof. The critical instant scenario is constructed as in Theorem 7. Under FIFO scheduling, the actual interference experienced by the q-th activation of task i depends on its arrival. Hence, we would need to evaluate multiple critical instant scenarios (cf. Theorem 14). Here, however we are only interested in a conservative upper bound of the multiple-event queueing delay. By conservatively assuming that the q-th activation of task i arrives just at the end of the of its queueing delay, we can reduce the number of critical instant scenarios to a single one [15]. Note that this arrival might occur later than the bound provided by the maximum distance function of task i's input event model. The q-th activation of task i can be executed when its $q-1$ predecessors have completed their execution (first term) and all activations of its interfering tasks (i.e. tasks in $fifo(i)$) that arrived within the queueing delay of the q-th activation of task i have completed execution (second term). Note, like in Theorem 9, we have to use the closed-interval version of the upper arrival function in order to handle the case that an interfering task is activated just at the end of the queueing delay. □

With the help of Theorem 8, Eq. (3.27) can be solved by iteration.

Theorem 13. *The multiple-event processing time for the q-th activation of task i under FIFO scheduling is upper bounded by*

$$B_i^+(q) = Q_i(q) + C_i^+ \tag{3.28}$$

Proof. Follows directly from Definitions 20 and 16. □

As mentioned before, under FIFO scheduling the interference experienced by the q-th activation of a task i during its multiple-event processing time depends on when the event that leads to this activation arrives. If the event q arrives early (within the bounds of its event model), it might experience less interference from its FIFO interferers, but more interference from (some of) its its own $q-1$ queued (i.e. unprocessed) predecessors. If, on the other hand, it arrives late (within the bounds of its event model), it might experience more

interference from its FIFO interferers, but less interference from its its own predecessors.

In Eqs. (3.27) and (3.28) we deliberately derived upper bounds on the multiple-event queueing delays and processing times that are independent of the event arrivals of task i. This has the benefit that Eqs. (3.27) and (3.28) can be used directly within the analysis framework described in Section 3.3.1. The worst-case response time of the q-th activation, for example, can be upper bounded by Eq. (3.12). However, as [50] and [15] demonstrate, possibly tighter bounds can be derived by investigating individual event arrivals, i.e. by evaluating multiple critical instant scenarios that differ in the arrival time of the q-th activation of task i.

Theorem 14. *The multiple-event queueing delay under FIFO scheduling for the q-th activation of task i that arrived at time a relative to the beginning of the multiple-event queueing delay is upper bounded by*

$$Q_{i,a}(q) = (q-1)C_i^+ + \sum_{j \in fifo(i)} \eta_j^{+]}(a)\, C_j^+ \tag{3.29}$$

Proof. The proof is identical to the proof of Theorem 12 with the exception that the event arrivals of interfering tasks only need to be considered up to the arrival of the q-th event of task i at time a. □

With the help of Theorem 8, Eq. (3.29) can be solved by iteration.

This improved multiple-event queueing delay can then be used to derive tighter bounds on the worst-case response time of the q-th event of a task i.

Theorem 15. *Under FIFO scheduling, the worst-case response time of the q-th activation of task i that arrived at time a is upper bounded by*

$$R_i(q) = \max_{a \in A_{q,i}} \left\{ Q_{i,a}(q) + C_i^+ - a \right\} \tag{3.30}$$

where $A_{q,i}$ is the set of arrival times to be evaluated for the q-th activation of task i.

Proof. Follows the argumentation of Theorem 13 and Eq. (3.12) with the exception that, instead of the earliest arrival time of the q-th event of task i, i.e. $\delta_i^-(q)$, the arrival time of the investigated candidate, i.e. a, is subtracted from the multiple-event processing time $Q_{i,a}(q) + C_i^+$. □

The set of arrival times to be evaluated for the q-th activation of task i can be constrained.

Theorem 16. *The set $A_{q,i}$ of arrival times to be evaluated for the q-th activation of a task i can be limited to the times where a coincides with the arrival times $\delta_j^-(n)$ of interfering task activations:*

$$A_{q,i} = \bigcup_{j \in fifo(i)} \left\{ \delta_j^-(n) \,|\, \delta_i^-(q) \leq \delta_j^-(n) < S_i(q) \right\}_{n \geq 1} \tag{3.31}$$

Proof. See [15] for a proof. The main argumentation is as follows. The earliest time the q-th activation of task i can arrive is bounded by its minimum distance function $\delta_i^-(q)$. An upper bound for the latest arrival of the q-th activation of task i is its scheduling horizon.

Furthermore, the candidates in $A_{q,i}$ are used to compute the worst-case response time in Eq. (3.30). This equation is the difference between a step function $Q_{i,a}(q)$, which steps (if it steps) in a and a linear monotonical increasing function $a - C_i^+$. It is trivial to see that this difference is maximal when the function steps, i.e. at a. Hence it is sufficient to only evaluate Eq. (3.30) at discrete times $a \in A_{q,i}$.

□

The worst-case response time (over all activations) of task i can then be computed as in Eq. (3.13).

3.3.2 System-Level Analysis

Resource-level analysis only considers resources in isolation. Actual systems, however, especially distributed systems such as networks, usually comprise multiple resources with many communicating tasks. These tasks often have complex functional and non-functional dependencies. Functional dependencies are a result of communication dependencies (e.g. along a task chain), while non-functional dependencies are introduced by resource sharing, e.g. interference from other tasks.

The goal of the system-level analysis is to combine the results of the independent resource-level analyses and their dependencies in a compositional way to derive system-level properties. Event models are used as interfaces to link the resource-level analyses via their abstract communication patterns.

The system-level analysis is performed iteratively. In each step, resource-level analyses are run in isolation until the system reaches a fixed point (steady state). A formal proof of the compositional performance analysis approach (e.g. existence of a fixed point) and the constraints on the design of resource-level analyses is presented in [161].

3.3.2.1 System-Level Analysis Flow

Figure 3.2 gives an overview of the system-level analysis flow. The system-level analysis starts by initializing the input event models of all tasks. For tasks without predecessors, environment event models must be supplied, that model the timing behavior of external stimuli, e.g. timer interrupts, (periodic) sensor or video samples, etc. At this stage the event models of all dependent tasks are unknown, as no resource-level analysis has been performed. These event models are initialized by optimistic guesses, typically by copying the environment event model to all tasks along its dependency tree.

Then a resource-level analysis is performed for each resource in isolation. This analysis derives bounds on the timing behavior of its tasks from which new output event models are derived (cf. Section 3.3.1.4.3).

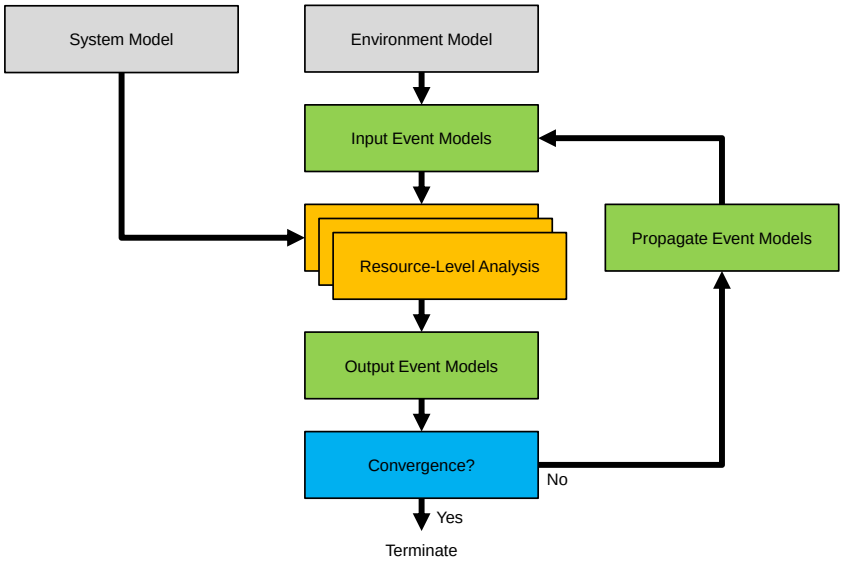


Figure 3.2: Compositional performance analysis flow

The system-level analysis then checks if all output event models are stable, i.e. did not change with respect to the previous iteration. If this is the case, the system-level analysis has reached a fixed point [161] and terminates. System-level properties can then be derived (see Section 3.3.2.2). Otherwise the output event models of all tasks are propagated as the new input event models to their dependent tasks and the system-level iteration is repeated.

Optionally, at the end of each system-level or resource-level analysis step, the analysis can check whether certain properties (e.g. worst-case response times, jitter, or activation backlogs) still meet (predefined) constraints and abort the analysis otherwise, reporting the violation.

3.3.2.2 System-Level Analysis Results

If the system-level analysis terminates, all event models are stable. This implies that also the results from the resource-level analysis have become stable and system-level metrics can be derived by combining these resource-level analysis results.

3.3.2.2.1 End-to-End Latency Bounds

The end-to-end latency along a path is an important metric in distributed real-time systems, as data may pass multiple ECUs, gateways, buses, or network switches on its way from its source to its destination. From a modeling perspective, a path can be abstractly defined as follows.

Definition 21 (Path)**.** *A path i is a chain of dependent[5] tasks $Path(i)$, such that the termination of one tasks produces an event that leads to the activation of its dependent (successor) task.*

We denote the first task in a path i, i.e. the first element of the task chain $Path(i)$, by $first(i)$. Likewise, we denote the last task by $last(i)$.

An upper bound on the worst-case response time along a path i can be obtained by summing the worst-case response times R_j^+ of each task $j \in Path(i)$.

Sometimes the data transported along a path is fragmented over multiple messages, the transmission of each of which is triggered by a separate event. In the Ethernet context, this is, for example, the case when IP fragmentation [89] is used or when a large message is spread across multiple TCP segments. In [49] it has been shown how an upper bound on the latency of multiple consecutive events along a path can be computed.

Theorem 17 (Maximum Multiple-Event Path Latency)**.** *An upper bound on the maximum latency of q consecutive events along a path i, starting with the first activation $first(i)$ and ending with the completion of the q-th activation of $last(i)$, is:*

$$l_i^+(q) = \delta^-_{first(i)}(q) + \sum_{j \in Path(i)} R_j^+ \tag{3.32}$$

where $\delta^-_{first(i)}(q)$ is the minimum distance function of $first(i)$.

Proof. An intuitive proof is given in [45]. We only provide a summary for completeness. In order to transmit q messages, these messages must be triggered by q events, leading to q activations of all tasks in $Path(i)$. We conservatively assume that the last message experiences worst-case delays (response times) at each resource. Since events are processed in-order, when the q-th activation of $last(i)$ completes, all other activations must have completed as well. □

Note that, since $\delta^-(1) = 0$, Eq. (3.32) includes the aforementioned path latency for a single event, i.e. the sum of worst-case response times, as a special case.

3.4 Summary

In this chapter, we recapitulated the compositional performance analysis approach for formal timing verification of complex systems. This approach can be used to derive best-case and worst-case bounds on timing properties (which translate into communication latencies in Ethernet networks) as well as worst-case bounds on buffer sizes.

[5] Note that the requirement of dependent tasks can be generalized to also cover semantically dependent tasks, i.e. paths where not all tasks are activated by their predecessors, but still exhibit task chain semantic. Examples include register communication with periodic sampling. In these cases, a sampling delay must be considered when computing end-to-end latency bounds [56].

First, we gave an overview of the CPA system model, which comprises a structural model (to abstract the hardware platform and the software components running on it) and a timing model (to abstract the timing of software components and their interaction). Then, we summarized how a timing analysis can be conducted based on this system model. This analysis comprises a compositional system-level analysis, which, by fixed point iteration, combines the outputs of resource-level analyses.

In the following chapters, we will extend the CPA framework towards the timing analysis of automotive Ethernet networks.

Chapter 4

Compositional Performance Analysis of Ethernet

As established in Chapter 3, formal performance analysis methods are a key tool to understand a system's worst-case behavior. This especially applies to safety-critical automotive systems connected via Ethernet.

Ethernet TSN [85] introduces different transmission selection mechanisms for Ethernet frames queued at switch ports, each with different design goals. If these mechanisms are to be used in safety-critical systems it is of utmost importance to understand their worst-case behavior. This will help us to identify and understand for what use cases a particular transmission selection mechanism is best suited.

This chapter contributes two key parts. First, in order to be able to apply CPA, we show how Ethernet network components can be mapped to the CPA system model. This allows us to reuse CPA's system-level analysis so that we can focus on developing resource-level analyses for each Ethernet TSN standard. Second, we present these resource-level analyses for the transmission selection mechanisms from different Ethernet standards. We start with the static priority based IEEE 802.1Q and continue to transmission selection mechanisms of the Ethernet TSN standards. Namely, we focus on IEEE 802.1Qbv, which enables scheduled frame transmission for low latency traffic, IEEE 802.1Qch, which aims to reduce the complexity of latency computation, and IEEE 802.3br, which introduces frame preemption to Ethernet in order to reduce blocking effects and to increase the link utilization of IEEE 802.1Qbv.

This chapter is based on [171] and [169].

4.1 Related Work

Due to its wide-spread adoption as a communication network in various domains, there exits a vast amount of related work on formal Ethernet timing analysis approaches.

4.1.1 IEEE 802.1Q

Ethernet without priorities has been formally analyzed by [117]. In [147] an analysis of static priority Ethernet using real-time calculus is presented. This analysis however shows large overestimations. A CPA-based analysis of switched Ethernet has been presented in [150].

As many industrial use cases require more stringent latency guarantees and, hence, thorough latency control, both standardization as well as formal analyses evolved. Ethernet AVB (IEEE802.1Qav) has become an official amendment to IEEE 802.1Q with the 2014 release of this standard. A formal analysis of Ethernet AVB is presented in [88]. This analysis, however, is limited to per-class timing properties and, hence, cannot be used to derive per stream timing guarantees. A CPA-based Ethernet AVB analysis is presented in [47] and [50], where the former introduces a CPA system model for the formal timing analysis of Ethernet networks (which we will use later in this chapter) and the latter the actual analysis. This analysis allows the derivation of per stream timing guarantees. The authors compare Ethernet AVB's CBS to plain IEEE 802.1Q and further show that the end-to-end latencies of shaped traffic classes (i.e. AVB class A and B) suffer significantly from AVB's traffic shaper design. Increasing the credit of AVB's traffic shaper beyond the actual required bandwidth (overreservation) helps to mitigate this problem [50]. The analysis from [50] has been extended by [17] to exploit the effects of AVB's traffic shaper to derive tighter timing guarantees. This optimization is limited to networks in which all links run at the same speed and only exploits the shaping of different traffic classes on the same link in isolation, neglecting the fact that all traffic that passes a link is correlated (i.e. shaped by the link's speed). A different busy period based analysis is presented by [27]. It was initially limited to networks containing just a single (central) switch, rendering it insufficient for typical automotive networks, but has since been extended by [13] to support multi-hop networks. In [145], an Ethernet AVB analysis based on real-time calculus is presented.

4.1.2 IEEE 802.11Qbv and IEEE 802.1Qch

With its gate control lists, IEEE 802.1Qbv defines a very general time-triggered transmission selection mechanism. This allows two typical integration modes when using IEEE 802.1Qbv: (1) A slot of fixed length is scheduled periodically for each scheduled traffic class. All frames of the corresponding traffic class use this slot and, should the situation arise, compete for link access. Due to this, a formal performance analysis for scheduled traffic is required to derive the timing properties of individual scheduled traffic streams. (2) Each traffic stream is scheduled in its own dedicated slot. This isolates individual scheduled traffic streams (within the same traffic class) and is beneficial if many traffic streams with different periods exists (that are not integer multiples of each other). With this approach, the timing of the individual scheduled traffic streams is known (and guaranteed) by design with the drawback of reduced link utilization, due to the presence of guard bands.

A first attempt of a formal timing analysis of TSN's traffic shapers has been presented in [166]. This analysis, however, has two main limitations: (1) For both IEEE 802.1Qbv and IEEE 802.1Qch, only a single frame per traffic class is considered and this frame is annotated with a static forwarding time through a switch. I.e. interference by same-priority frames (frames of equal priority competing for link access) is not considered. While this corresponds to integration mode 2 for IEEE 802.1Qbv, this is a major drawback for IEEE 802.1Qch, as the limited number of Ethernet priorities will require priority sharing in realistic setups and, hence, will introduce same-priority interference. (2) Higher-priority interference is not fully considered for IEEE 802.1Qbv and IEEE 802.1Qch, i.e. it is only possible to analyze a single scheduled or cyclic traffic class, which must be on the highest priority. Actual systems, however, might have more than one cyclic traffic class, which might not be on the highest priority [105].

This thesis, in contrast, presents a formal analysis based on the proven CPA framework (elaborating on the author's own work in [171]), which does not suffer from these limitations and, hence, allows the formal analysis of all traffic streams across all traffic classes, including same-priority interference and multiple cyclic traffic classes. This analysis supports both integration mode 1 and 2. After the publication of [171], further formal performance analyses of IEEE 802.1Qbv have been presented.

In [54], a CPA-based analysis is presented that can, like [171], also be applied to arbitrary gate control lists. However, [54] only investigates the impact of scheduled traffic on non-scheduled traffic, i.e. assuming integration mode 2. A formal performance analysis of IEEE 802.1Qbv that is based on network calculus is presented in [108]. This analysis does not impose any constraints on the gate control list, i.e. it allows for interference of scheduled traffic streams within a shared slot (integration mode 1) and even allows scheduled slots of different traffic classes to overlap. In [182] a network calculus based analysis is presented that exploits correlations of aggregated traffic streams (due to shared links and due to known offsets between gate control list schedules of consecutive hops), reducing analysis pessimism significantly. As rather generic analysis approaches, [108] and [182], do not require synchronized (i.e. scheduled) end stations.

An alternative to a worst-case performance analysis for scheduled traffic in IEEE 802.1Qbv is to explicitly design gate control list schedules so that, by construction, all timing requirements of scheduled traffic are met (cf. integration mode 2). This, however, is a non-trivial problem [162] and various heuristic approaches have been proposed to find solutions to this schedule synthesis problem, e.g. [40], [157], [51], [140] (the latter does also include a heuristic to find paths for AVB traffic streams). In [126] the authors consider entire platforms with communicating tasks and end-to-end task chain timing requirements. They propose a heuristic approach to jointly consider the mapping of tasks to the platform's processing cores and the generation of gate control lists to allow these tasks to communicate while meeting their timing requirements.

As these works focus solely (or primarily in case of [140]) on scheduled traffic, less-critical, non-scheduled traffic (such as AVB traffic) may suffer unnecessarily. This problem is addressed by [61] by also taking the timing requirements of additional AVB traffic streams into account to guide the search

heuristic. The work in [63] jointly considers the AVB-aware gate control list synthesis (including support for multiple scheduled traffic classes) and the problem of finding paths for each scheduled traffic flow through the network. In [62] mixed-critical communication is classified into hard real-time, soft real-time, and best effort traffic. The authors propose a heuristic approach to assign hard and soft real-time traffic to scheduled and AVB traffic classes and also to derive the corresponding configuration parameters (gate control lists, AVB's CBS slopes), ensuring that hard real-time traffic meets its deadlines and the utility of soft real-time traffic (measuring the nearness to the (soft) deadline in cases of exceedance) is maximized.

Relaxing the explicit gate control list schedule planing by removing constraints on when to transfer frames inside a scheduled interval and allowing scheduled intervals to overlap, some degree of implementation freedom can be gained at the price of (again) needing a formal analysis to ensure schedulability.[1] Such an approach, based on the analysis of [108], is discussed in [146].

The key challenges of IEEE 802.1Qch are the dimensioning of alternating intervals and the mapping of traffic streams to these intervals. While the work presented in this thesis can be seen as a worst-case analysis that does not make any assumptions on when frames of traffic streams are injected into the network (and into the IEEE 802.1Qch interval schedule, in particular), later publications take a synthesis-based approach and provide methods to create feasible stream to interval mappings, assuming that the frame injection times can be controlled. Particularly, in [181], the authors propose to plan the frame injection times, i.e. to which of the intervals frames of a traffic stream are mapped, so that, by design, their timing requirements are met. They formulate the frame injection time planing problem as a network-wide optimization problem and use heuristics guided by constraints and domain-specific knowledge to solve it. In doing so, they ensure (and, at the same time, exploit) that IEEE 802.1Qch's property of the end-to-end latency of a traffic stream being only a function of the number of traversed switches is met.

4.1.3 IEEE 802.3br

A simulation-based evaluation of IEEE 802.3br is presented in [98]. In [106], an experiment-based evaluation of a custom preemption mechanism, which has a slightly larger overhead than IEEE 802.3br, is presented. Both [98] and [106] confirm that frame preemption reduces the latency and jitter of high-priority traffic. However, no formal worst-case guarantees are given.

An early academic version of TTEthernet supported frame preemption. However, preempted frames were not resumed but retransmitted, resulting in poor link utilization. Commercial TTEthernet implementations only support timely blocking (block non-critical frames, so that there is no overlap with time-triggered critical frames) and shuffling (accept delay from non-preemptiveness)

[1] In [65], a simulation-based comparison between different constraints of configuring scheduled traffic (explicitly scheduled per frame versus a more relaxed mapping of traffic streams to scheduled traffic classes) is presented.

[14]. In [129], an actual frame preemption mechanism for TTEthernet is presented. The evaluation, however, is only simulation-based and does neither consider latency nor jitter. Only frame drop rates are investigated.

Frame preemption has also been evaluated in the AFDX context [58]. Here, however, the authors also only consider the complete retransmission (and not the resumption) of a frame after it has been preempted.

In thesis we present and elaborate our work done in [169], which appears to have been the first formal performance analysis on IEEE 802.3br. After the publication of this work, the formal analysis of IEEE 802.3br gained more attention from the research community. In [109], the authors also investigate the effect of frame preemption on AVB traffic that is scheduled below IEEE 802.1Qbv traffic. The benefit of frame preemption (a little less than 10 %) on AVB traffic is similar to the benefit of frame preemption on non-scheduled traffic we are observing in our analysis.

4.1.4 IEEE 802.1CB

Successfully managing and handling the redundancy introduced by IEEE 802.1CB is a non-trivial task. In [143], the authors outline challenges and limitations that are introduced by IEEE 802.1CB. In particular, buffer dimensioning for sequence recovery, implications of out-of-order frame delivery, increased interference, limitations of the error feedback mechanism, as well as (re)configuration complexity are highlighted, underlining the need for formal performance analysis methods in order to effectively deploy IEEE 802.1CB in (in-vehicle) networks.

The work in [176] presents such formal methods to analyze frame replication and elimination in time-sensitive networks in the network calculus framework, which can be used to analyse the timing impact of IEEE 802.1CB. It provides methods to derive tight output traffic characterizations after the frame elimination process. Furthermore, it also analyzes the interactions and implications of frame elimination and different kinds of traffic regulators (i.e. per-flow regulators and IEEE 802.1Qcr's asynchronous traffic shaper). One key outcome of [176] in the Ethernet TSN context is that, for the asynchronous traffic shaper, when presented with a stream of out-of-order frames (as it can occur in IEEE 802.1CB during the frame elimination process [78]), no formal delay bounds can be guaranteed.[2] This is of particular importance as Ethernet TSN does not standardize a packet reordering function. If, however, such a reordering function would be available, [176] also provides means to derive its parameters (timeout and buffer size).

4.1.5 Combination of Ethernet TSN Transmission Selection Mechanisms

Many application domains contain a mix of different traffic streams with different timing requirements. To accommodate these requirements, different transmission selection mechanisms must be combined in a single Ethernet network.

[2] Per-flow regulators only experience a bounded delay penalty.

With Ethernet TSN, a common setup is the combination of IEEE 802.1Qbv for time-critical traffic, Ethernet AVB's CBS for less critical traffic (often including multiple shaped traffic classes), and best effort traffic on a lower priority. In [109], under the assumption of integration mode 2, a formal performance analysis based on network calculus for two classes of AVB traffic in such a mixed setup is presented. The analysis in [184] supports an arbitrary number of AVB traffic classes. It also exploits correlations of aggregated traffic streams due to shared links and CBS shapers, which, again, helps to significantly reduce analysis pessimism. Additionally, it supports different kinds of credit replenishment behavior during the guard bands that protect scheduled traffic (frozen credit, as assumed by most analyses and non-frozen credit, as standardized).

In [130] a combination of latency rate limited control data traffic on the highest priority and Ethernet AVB's CBS is presented. AVB traffic is additionally regulated via IEEE 802.1Qcr's asynchronous traffic shaper before being enqueued into the CBS classes.

A comprehensive formal analysis based comparison of individual and different combinations of Ethernet TSN transmission selection mechanisms has been compiled in [183]. The evaluation is based on both synthetic (different sized ring, mesh, and tree topologies) as well as realistic use cases, using end-to-end latency bounds, backlog, and end-to-end jitter as the comparison metrics. It addresses two important aspects. First, it compares different transmission selection mechanisms in isolation. Second, it evaluates the combination of IEEE 802.1Qbv with SPNP and IEEE 802.1Qbv with CBS, both with and without IEEE 802.1Qcr's asynchronous traffic shaper. Special attention is paid to evaluate the conditions under which it is advantageous to deploy asynchronous traffic shapers. From the evaluation it appears that asynchronous traffic shaping is best applied when the load of the shaped traffic streams is high or when there is a certain amount or (higher-priority) interference on the shaped traffic streams. One important observation is that not all metrics evolved proportionally, e.g. the authors observed cases where the end-to-end latency bounds show smaller or even adverse impact while the backlog bounds showed significant improvements. This, once more, underlines the crucial need for formal analysis methods to find optimized solutions to the diverse communication architectures of mixed-critical embedded systems.

4.1.6 Burst-Limiting Shaper

In the context of Ethernet TSN, a burst-limiting shaper (BLS) has been discussed. The BLS is similar to Ethernet AVB's CBS. But, while CBS only allows bursts of critical traffic after blocking by interfering traffic, i.e. to help critical traffic to compensate for the lost blocking time, the design goal of the BLS, in contrast, is to allow bursts of limited size without prior blocking during normal operation. The second key differentiator between BLS and CBS is that BLS was envisioned to be non-blocking, i.e., during intervals of credit replenishment, a shaped traffic class would be mapped to a lower priority instead of being blocked.

The BLS never made it past the project authorization request (PAR) phase of the IEEE standardization process. Nevertheless, during the time of active

discussion of the BLS, formal analyses have been proposed. An informal discussion of the BLS's worst-case bounds was presented in [105]. A first attempt to formally analyze the timing behavior of the BLS was proposed in [166]. This analysis, however, has two main limitations: (1) It does not consider interference by same-priority traffic. (2) It does not consider higher-priority interference on BLS traffic, i.e. it is only possible to analyze a single BLS traffic class, which must be on the highest priority.

A CPA-based analysis attempt was presented in [170]. While addressing the limitations of [166], it makes the (over)simplifying assumption of treating BLS as a blocking shaper. It has been shown, however, in [57] that this can lead to optimistic results. Furthermore, [57] identified sources of unnecessary pessimism in the BLS analysis of [170].

The BLS was also evaluated in the AFDX community, where [12] and [11] proposed to extend AFDX to support two additional traffic classes (in addition to the existing rate-limited traffic class): (1) a high-priority class for safety-critical traffic and (2) a low-priority traffic class for best effort traffic. In order to bound the impact of the safety-critical traffic on the other classes, it is shaped by a BLS. Both analyses are based on network calculus. The work in [57] generalizes the analysis of [11]. It presents tightness and sensitivity analyses regarding traffic utilization rates of different traffic classes as well as key BLS parameters (i.e. upper and lower credit limits, and reserved bandwidth). These analyses are also performed with [12] and with the CPA approach from [170] with [57] performing best. Due to the lower complexity of the network calculus based analysis in [57], its analysis times are significantly shorter than the analysis times of [170].

4.1.7 AFDX

AFDX, as a communication backbone for avionic networks, has received significant attention in the context of formal performance analysis methods. AFDX analyses based on the busy period are presented in [150] and [64]. In [23] and [24], the trajectory approach [122] is adapted to the analysis of AFDX networks, including the effect of frame serialization. [23] also compares the trajectory approach to a network calculus implementation of the analysis and achieves, on average, 10% shorter end-to-end latency guarantees. The computation of frame backlog is presented in [25]. In [103] sources of optimism in the trajectory approach have been identified and subsequently fixed (partly) by [119]. One key limitation of the trajectory approach is that it can only derive upper bounds for the end-to-end latency of traffic streams, if the sum of all loads along the analyze trajectory is less than 100 % [60]. To overcome this limitation, a different analysis, called the forward end-to-end delays analysis has been proposed [104] and improved to exploit frame serialization in [60].

4.1.8 The Impact of Clock Synchronization on Formal Timing Analysis

Typically, formal analysis frameworks implicitly assume ideal (i.e. perfectly synchronized) clocks at all resources. In reality, however, this may not be

the case (especially for distributed systems), either because clocks are (intentionally) unsynchronized (e.g. to reduce attack surface) or can only be kept synchronized within certain bounds (e.g. via IEEE 802.1AS). The seminal work in [174], discusses this synchronization issue and introduces formal methods to support systems with multiple clocks (both unsynchronized and synchronized) by extending network calculus accordingly.

In particular, [174] focuses on per-flow and interleaved traffic regulators, the latter of which are relevant in the Ethernet TSN context in the form of IEEE 802.1Qcr's asynchronous traffic shaper. It is shown that asynchronous traffic shapers require adjustments of their parameters along the paths of traffic flows (essentially ensuring that, for each downstream shaper, the output rate is larger than input rate), regardless of whether it is being used in a network with unsynchronized or synchronized clocks. Otherwise it can induce unbounded delays on the traffic flows that it shapes.[3]

4.2 Ethernet System Model

In order to derive formal timing guarantees for Ethernet networks, Ethernet-specific constructs must be mapped to the abstract CPA system model entities. Following Section 3.2, we divide the Ethernet system model into a structural model and a timing model.

4.2.1 Ethernet Structural Model

Ethernet networks comprise end stations, which produce and consume Ethernet frames, and switches, which forward Ethernet frames. The network's topology defines how end stations and switches are interconnected. Traffic streams are used to model the transmission of Ethernet frames through the network. In switched networks, congestion (between different traffic streams) happens at the switches, where it is resolved by transmission selection mechanisms (i.e. scheduling mechanisms).

In this section, we argue which of these Ethernet-specific constructs are modeled by which entities of CPA's structural model.

4.2.1.1 Platform

The topology of an Ethernet network is mapped to a platform in the CPA structural model.

4.2.1.2 Resources

In the CPA model, resources are points of contention at which (potentially) multiple tasks concur for service. In Ethernet, switches are the points of contention as here the frames of multiple traffic streams concur for resources

[3] Per-flow regulators are less sensitive than the interleaved regulators of IEEE 802.1Qcr, but in some cases also require parameter adjustments [174]. Their implementation effort, however, is currently considered to be too high for a hardware implementation [159].

and (ultimately) link access. As discussed in Section 2.3.1, inside an Ethernet switch, there are multiple points of contention and delay.

At the input ports of some switch architectures, contention can lead to input queueing delay. However, shared memory switches typically do not suffer from input queueing delay. Furthermore, evaluating the frame check sequence and performing ingress filtering and policing (e.g. via IEEE 802.1Qci) may introduce a certain delay. This delay is implementation dependent.

The switch fabric introduces fabric delay, since frames must be passed from input to output ports. In shared memory switches, for example, this involves writing frames to and reading frames from memory. If the switch fabric cannot serve all input and output ports of the switch at line speed, then it will also be a point of contention, introducing additional delay. Note that, in modern shared memory switches, the switch fabric is sufficiently fast so that only a small fabric delay is introduced. This delay is implementation dependent.

In the output ports of switches, there is output queueing delay. Here, a transmission selection mechanism arbitrates link access among the queues of different Ethernet traffic classes. The output queueing delay is the time a frame has to wait until it is selected for transmission (cf. Definition 20). In our switch model the output ports are the main points of congestion.

Lastly, there is a transmission delay through the PHYs (from MII to MDI and vice versa) and over the medium. This is the time it takes to propagate data from one switch port to its peer's switch port. The transmission delay through the PHYs is technology-dependent and shall be about 1 us (sum of delays on sender and receiver PHYs) for typical automotive PHYs [29], [136]. The transmission delay on the medium can be bounded by the speed at which electrical signals propagate on a wire. Assuming a maximum wire length of 15 m, which is a typical design criterion for automotive Ethernet (see Section 2.1.2), the PHY transmission delay is the dominating term and we assume an overall transmission delay of about 1 us.

We conclude that modern switches are fast enough to eliminate contention in the ingress ports and switch fabric. Implying that the input (queueing) delay and the fabric delay are typically reduced to delays in the order of a few clock cycles and/or can be assumed constant. Hence, their timing impact can be added during the end-to-end latency computation (cf. Eq. (3.32)). Similarly, the transmission delay per link is also added to the end-to-end latency. Thus, we only model the switch output ports, as the points of contention, by CPA resources.[4]

Following the same argumentation, the timing behavior of end stations can similarly be modeled by their output ports.

4.2.1.3 Scheduler

In CPA, a scheduler defines how access to a resource is arbitrated. In the Ethernet context, a scheduler comprises the queueing and transmission selection mechanisms in the output ports. Inside the queues of an output port, frame

[4]Note that the generic CPA model still allows to also include more detailed resource models of, for example, the input ports or the switch fabric if required.

of identical traffic class interfere with each other, while, in the transmission selection stage, frames of different traffic classes interfere (cf. Section 2.2.1.3).

4.2.1.4 Tasks

In the CPA model, tasks are entities that consume service from resources. In the Ethernet context, the transmission of a frame at a switch output port is modeled by the execution of a task on the resource modeling this output port.

4.2.1.5 Applications

In CPA, an application is defined as a set of interconnected tasks, forming a task graph. In the Ethernet context, we us these applications to model traffic streams. A traffic stream is a sequence of semantically related frames from a source to one or more destination(s). On each switch output port (CPA resource) that is traversed by the traffic stream, a task is used to model the transmission of the stream's frames. One task execution corresponds to the transmission of a single frame. These task are interconnected according to the traffic stream's path through the network. Note that this path may include forks when the traffic stream has more than one destination, i.e. when it is a multicast or broadcast stream.

4.2.2 Ethernet Timing Model

The Ethernet timing model is used to reason about the timing behavior of Ethernet frames.

4.2.2.1 Execution Times

Ethernet frames require a certain time, called the transmission time, to be sent over a link. The transmission time is modeled by the execution time of a task in the CPA model and depends on the frame's size and the transmission rate of the output port that transmits the frame.

Definition 22 (Port Transmission Rate)**.** *Let r_{TX} be the transmission rate of an output port.*

In the context of this thesis r_{TX} is given in bytes/second unless stated otherwise. Often, the port transmission rate is also called port speed.

Like the task execution time, the frame transmission time does not include any interference from other frames. Since a frame's payload may vary, we define lower and upper bounds on its transmission time.

Definition 23 (Frame Transmission Time Bounds)**.** *The minimum and maximum frame transmission time bounds of a frame of a traffic stream i with a minimum payload of p_i^- bytes and a maximum payload of p_i^+ bytes at a switch*

port with a transmission rate of r_{TX} bytes/second in the absence of any interference are:

$$C_i^- = \frac{42\,\text{bytes} + \max\left\{42\,\text{bytes}, p_i^-\right\}}{r_{TX}} \tag{4.1}$$

and

$$C_i^+ = \frac{42\,\text{bytes} + \max\left\{42\,\text{bytes}, p_i^+\right\}}{r_{TX}} \tag{4.2}$$

where the first 42 bytes are composed of the Ethernet frame's preamble (7 bytes), start of frame delimiter (1 byte), destination and source addresses (2x 6 bytes), IEEE 802.1Q tag (4 bytes), EtherType (2 bytes), frame check sequence (4 bytes), and interframe gap (12 bytes) (cf. Sections 2.1.4 and 2.2.1). The second 42 bytes along with the max-operator ensure that the minimum Ethernet frame size requirement of 84 bytes is met (cf. Section 2.1.4 and Figure 2.4).

4.2.2.2 Events

The arrival and transmission of individual Ethernet frames are modeled by events.

4.2.2.3 Event Streams

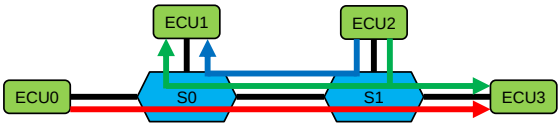
The semantically related frame arrivals and transmissions of an Ethernet traffic stream are modeled by event streams. The timing behavior of these frame arrivals and transmissions is bounded by event models.

4.2.3 Example

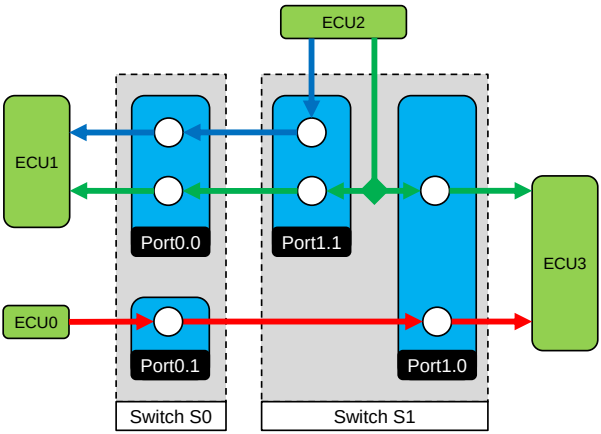
Figure 4.1a shows an Ethernet model of a network comprising four ECUs[5] and two switches. There are three traffic streams in the network: one unicast stream (red) from ECU0 to ECU3, one unicast stream (blue) from ECU2 to ECU1 and one multicast stream (green) from ECU2 to ECU1 and ECU3.

Figure 4.1b shows the corresponding CPA system model. The platform is represented by the entire figure, describing the interconnections of ECUs and switches. Each switch output port is modeled by a separate resource. Unused output ports (e.g. from switch S0 to ECU0) do not need to be modeled. On each output port, that is traversed by a traffic stream, a task models the transmission of the stream's frames, e.g. the red unicast stream from ECU0 to ECU3 passes the output ports from switch S0 to switch S1 and from switch S1 to ECU3, both of which have one task allocated to this stream. There are three applications, one for each traffic stream: (1) the application represented by the task graph containing red links for the unicast traffic stream from ECU0 to ECU3, (2) the application represented by the task graph containing blue links for the unicast traffic stream from ECU2 to ECU1, and (3) the application represented by the task graph containing green links for the multicast traffic stream from ECU2 to ECU1 and ECU3.

[5] In the automotive context end stations are typically electronic control units (ECUs).



(a) Ethernet network model



(b) CPA system model

Figure 4.1: Example mapping from the Ethernet model to the CPA system model

4.2.4 Notation Conventions

In the rest of this thesis, we adopt our notation to the Ethernet context and use terms that indicate the forwarding of frames. A *traffic stream* passing a network resource, such as a (switch) port, is modeled as a *task* and the *transmission of a frame of (traffic) stream* i becomes the *execution of task* i. Furthermore, we will use q*-th frame arrival of stream* i instead of q*-th activation of task* i and *forwarding time* instead of *response time*.

In the output ports of a switch or end station, Ethernet priorities (indicated by the PCP field in the Ethernet frame, cf. Section 2.2.1) are mapped to traffic classes. Each traffic class is associated with a queue and these queues are then served by the port's transmission selection mechanism (cf. Section 2.2.1.3). Multiple Ethernet priorities may share the same traffic class and, hence, queue. In our analysis this can be modeled by mapping the traffic streams, that share a traffic class, to the same priority. For the sake of simplicity, however, we assume that each Ethernet priority is assigned to its own traffic class/queue. This way we keep the argumentation from becoming overly complicated, i.e. we can reason about *frames from traffic streams with a higher priority than stream* i instead of *frames from traffic streams that are mapped to traffic class with a higher priority than the traffic class that traffic stream* i *is mapped to*.

Often, we need to reference the traffic class, that a traffic stream is mapped to.

Definition 24 (Traffic Stream to Traffic Class Mapping)**.** *Let the function* $cl(i)$ *map a traffic stream* i *to its corresponding traffic class.*

4.3 Analysis of IEEE 802.1Q

IEEE 802.1Q defines a priority-based transmission selection mechanism together with per traffic class queuing (cf. Section 2.2.1.3). The transmission selection mechanism is implemented by a SPNP scheduler. While the queuing mechanism is (intentionally) not specified in IEEE 802.1Q (except for an ordering constraint, see Section 2.2.1.3), it is typically implemented as FIFO queueing. This is also the queuing mechanism we assume in this thesis.

4.3.1 Resource-Level Analysis of IEEE 802.1Q

In order to integrate the performance analysis of IEEE 802.1Q into the resource-level analysis framework outlined in Section 3.3.1, we need to define and compute the multiple-event scheduling horizon (as the stopping condition) and the multiple-event processing time as well as the multiple-event queueing delay in order to derive timing properties.

As discussed, IEEE 802.1Q combines SPNP and FIFO scheduling. Consequently, we can derive a resource-level analysis by combining the SPNP and FIFO analyses from Sections 3.3.1.5.1 and 3.3.1.5.2. To improve readability, we split the blocking effects in multiple disjunct terms, each focusing on a specific kind of blocking. Overall, we identify three different kinds of blocking: lower-priority blocking, same-priority blocking, and higher-priority blocking.

The lower-priority blocking corresponds to the lower-priority blocking term from Eq. (3.24).

Lemma 1 (Lower-Priority Blocking in IEEE 802.1Q)**.** *The lower-priority blocking a traffic stream* i *can experience in IEEE 802.1Q is bounded by*

$$I_i^{LPB} = \max_{j \in lp(i)} \left\{ C_j^+ \right\} \tag{4.3}$$

where $lp(i)$ *is the set of all traffic streams with a priority lower than that of stream* i*.*

Proof. In non-preemptive scheduling, the transmission of a frame of stream i can be blocked by at most one lower-priority frame, if this lower-priority frame starts transmitting just before the first frame of stream i arrives. In the worst-case, this is the longest lower-priority frame (cf. Theorem 9). □

The same-priority blocking can be modeled by a FIFO scheduler (cf. Section 3.3.1.5.2).

Lemma 2 (Same-Priority Blocking in IEEE 802.1Q)**.** *The same-priority blocking the q-th frame of a traffic stream i that arrived at time a can experience in IEEE 802.1Q is bounded by*

$$I_i^{SPB}(q,a) = (q-1)C_i^+ + \sum_{j \in fifo(i)} \eta_j^{+]}(a) C_j^+ \tag{4.4}$$

where $fifo(i)$ is the set of FIFO interferes of traffic stream i. In the Ethernet context, this is the set of all traffic streams that are mapped to the same queue as traffic stream i, i.e. the set of all traffic streams with a priority equal to that of stream i (excluding stream i).

Proof. The q-th arrival of a frame of stream i, which arrived at time a, has to wait for its own $q-1$ predecessors to finish and for all frames of other same-priority traffic streams, which have arrived previous to its arrival. The amount of same-priority blocking is maximized, if the critical instant scenario is chosen such that frames of same-priority traffic streams start arriving as fast as possible together with the first frame of traffic stream i. In the worst-case, frames of same-priority traffic streams might arrive just at the end of a. We include the interference from these frames by using the closed-interval upper arrival function, i.e. assuming that these frame arrive just before the q-th frame of stream i at a. □

The higher-priority blocking corresponds to the higher-priority blocking term from Eq. (3.24).

Lemma 3 (Higher-Priority Blocking in IEEE 802.1Q)**.** *The higher-priority blocking a traffic stream i can experience in any time interval Δt in IEEE 802.1Q is bounded by*

$$I_i^{HPB}(\Delta t) = \sum_{j \in hp(i)} \eta_j^{+]}(\Delta t) C_j^+ \tag{4.5}$$

where $hp(i)$ is the set of all traffic streams with a priority higher than that of stream i.

Proof. See Theorem 9. □

Now, the multiple-event queueing delay can be derived.

Theorem 18. *The multiple-event queueing delay in IEEE 802.1Q for the q-th frame arrival of traffic stream i that arrived at time a is upper bounded by*

$$Q_{i,a}(q) = I_i^{LPB} + I_i^{SPB}(q,a) + I_i^{HPB}\left(Q_{i,a}(q)\right) \tag{4.6}$$

Proof. The multiple-event queueing delay must take all blocking effects into account. From Lemmata 1, 2, and 3 the theorem follows. □

With the multiple-event queueing delay, the multiple-event processing time can be bounded.

Theorem 19. *The multiple-event processing time for the q-th frame arrival of traffic stream i in IEEE 802.1Q is upper bounded by*

$$B_i^+(q) = Q_i(q) + C_i^+ \tag{4.7}$$

where $Q_i(q)$ is an upper bound on the multiple-event queueing delay for the q-th frame arrival of traffic stream i in IEEE 802.1Q:

$$Q_i(q) = I_i^{LPB} + I_i^{SPB}\left(q, Q_i(q)\right) + I_i^{HPB}\left(Q_i(q)\right) \tag{4.8}$$

Proof. Follows the proofs of Theorems 9 and 12. □

Note that, in this case, we are only interested in time interval between the arrival of first and the completion of q-th frame transmission (cf. Definition 20). Consequently, we do not need to consider individual frame arrivals, but can safely assume that (in the worst-case) all frames arrive as soon as possible, i.e. substitute a by $Q_i(q)$.

With the help of the blocking terms in Lemmata 1, 2, and 3, also, the multiple-event scheduling horizon can be derived.

Theorem 20. *The multiple-event scheduling horizon for the q-th frame arrival of traffic stream i in IEEE 802.1Q is upper bounded by*

$$S_i(q) = I_i^{LPB} + \tilde{I}_i^{SPB}(q, S_i(q)) + \tilde{I}_i^{HPB}\left(S_i(q)\right) \tag{4.9}$$

with

$$\tilde{I}_i^{SPB}(q, \Delta t) = qC_i^+ + \sum_{j \in fifo(i)} \eta_j^+(\Delta t) C_j^+ \tag{4.10}$$

and

$$\tilde{I}_i^{HPB}(\Delta t) = \sum_{j \in hp(i)} \eta_j^+(\Delta t) C_j^+ \tag{4.11}$$

Proof. The scheduling horizon of the q-th frame arrival of traffic stream i ends, when all q frames of stream i have been transmitted and all frames of interfering traffic streams that arrived within the scheduling horizon of the q-th frame arrival of traffic stream i have been transmitted as well. Only then an additional hypothetical activation of infinitesimal small execution time could be processed. The amount of frames from interfering traffic streams can be bounded by Lemmata 1, 2, and 3. However, we have to adjust the blocking terms from Eqs. (4.4) and (4.5) with Eqs. (4.10) and (4.11) (respectively) to address the fact that the scheduling horizon is defined over a right half-open interval, i.e. we use $\eta_j^+(\Delta t)$ instead of $\eta_j^{+]}(\Delta t)$. Additionally, we need to consider all q frame arrivals when computing the multiple-event scheduling horizon for the q-th frame arrival of traffic stream i. This is also addressed by Eq. (4.10). □

Now, we can derive bounds of timing corner-cases and the number of backlogged frames, which can be transformed into a maximum buffer size requirement.

4.3.2 Worst-Case Frame Forwarding Times

The frame forwarding time corresponds to the response time (cf. Definition 18). Hence, substituting $Q_{i,a}(q)$ in Eq. (3.30) by Eq. (4.6), yields an upper bound on the worst-case frame forwarding time for the q-th frame arrival of stream i. To determine the set of (frame) arrival candidates $A_{q,i}$, we can apply Eq. (3.31). The overall worst-case frame forwarding time for frames of stream i, can the be bounded by Eq. (3.13).

4.3.3 End-to-End Latency Bounds

The end-to-end path latency that Ethernet frames experience on their way from a source end station to a destination end station is an important metric in automotive Ethernet networks. It can be computed as described in Section 3.3.2.2.1, i.e. via the maximum multiple-event path latency in Eq. (3.32). Note that in case of IP fragmentation, the multiple-event property of Eq. (3.32) is of particular significance.

4.3.4 Maximum Buffer Size Bounds

As discussed in Section 2.3.5, (shared memory) switches typically allocate memory using fixed-sized pages. An upper bound on the buffer size requirement in bytes for each traffic stream on a given switch port can be derived from the activation backlog (Eq. (3.16)) by taking a page size of m bytes in to account.

Definition 25 (Paged Buffer Size Bound)**.** *An upper bound on the maximum buffer size in bytes of a traffic stream i with frames that have a maximum payload of p_i^+ bytes on a given switch port with a page size of m bytes is:*

$$\tilde{b}_i^{+,m} = b_i^+ \left\lceil \frac{22\,\text{bytes} + \max\left\{42\,\text{bytes}, p_i^+\right\}}{m} \right\rceil m \tag{4.12}$$

where b_i^+ is the activation backlog as described in Eq. (3.16). We assume that only the destination and source addresses (2x 6 bytes), the IEEE 802.1Q tag (4 bytes), the EtherType (2 bytes), the payload p_i^+, and the frame check sequence (4 bytes) need to be stored (cf. Definition 23).

An upper bound on the maximum buffer size per switch port can be computed by summing the per-stream buffer size bounds on a given port. Consequently, an upper bound on the maximum buffer size per switch can be derived by summing the buffer size bounds of all its ports. Often, switches support to partition their global buffer memory based on priority in order to provide freedom from (storage) interference. A per priority upper bound for such configurations can be computed by summing the buffer size bounds per priority.

Our analysis assumes the absence of frame drops. Hence, in order for the analysis guarantees to hold, it must be checked that the computed upper bounds stay below the switch's available (global or per priority) buffer memory size.

4.4 Analysis of IEEE 802.1Qbv

As discussed in Section 2.2.3.2, IEEE 802.1Qbv [83] uses time-driven scheduling to arbitrate link access between traffic classes. For each traffic class, the scheduler at an output port contains a transmission gate, which allows frames to pass when opened and blocks frames when closed. The times at which these gates open and close are programmable via a gate control list that is cyclically repeated. Gates of multiple traffic classes can be open concurrently. Then, link access is arbitrated according to the priority of these classes. To prevent frames of a traffic class from being transmitted after its gate closed, IEEE 802.1Qbv defines guard bands. From the start of a guard band until the gate is closed, no new frames of the corresponding class are allowed to start transmission. IEEE 802.1Qbv allows very flexible control over the times at which the transmission gates of each traffic class are opened and closed. It is, for example, possible to mix traffic from different traffic classes (priority-based) by opening and closing their corresponding transmission gates concurrently.

4.4.1 Resource-Level Analysis of IEEE 802.1Qbv

As with IEEE 802.1Q, in order to integrate the performance analysis of IEEE 802.1Qbv into the resource-level analysis framework outlined in Section 3.3.1, we need to define and compute the multiple-event scheduling horizon and the multiple-event processing time as well as as the multiple-event queueing delay.

IEEE 802.1Qbv suggests that each critical traffic class only has link access during special (scheduled) time intervals, i.e. the gate control list is programmed such that only during these intervals the traffic class' transmission gate is open and that it is closed otherwise [83]. Following [83], we also assume that the scheduled intervals of different critical traffic classes are non-overlapping and not necessarily consecutive (in the sense that two intervals of different critical classes do not need to be aligned back-to-back), so that every critical traffic class has exclusive link access during its intervals, i.e. without interference by higher- or lower-priority traffic. However, there can still be interference from same-priority traffic (in integration mode 1). Outside these intervals, the gates of all other (less-critical or non-critical) traffic classes are open, i.e. traffic streams from these other classes compete for link access according to their priority.

To formalize the resource-level analysis of IEEE 802.1Qbv, we introduce a set of definitions.

Definition 26 (ST Classes, ST Streams, and ST Frames)**.** *Let $\mathcal{S}$ be the set of all Ethernet traffic classes, which receive exclusive link access by IEEE 802.1Qbv. We will call traffic classes $I \in \mathcal{S}$ scheduled traffic (ST) classes and, correspondingly, their traffic streams ST streams and frames of these streams ST frames.*

Likewise non-ST classes, non-ST streams, and non-ST frames are defined for traffic classes $J \notin \mathcal{S}$.

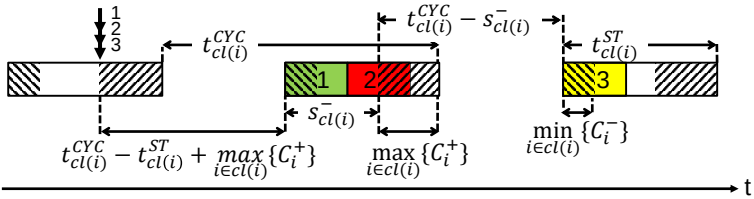


Figure 4.2: Illustration of the terms contributing to the closed-gate blocking time of the IEEE 802.1Qbv analysis for ST traffic streams

In the following, we assume that scheduled traffic for each $I \in \mathcal{S}$ is served periodically via a single interval during which its gates are exclusively open in the gate control list.[6]

Definition 27 (ST Interval and ST Cycle)**.** *Each ST class I is associated with a time interval t_I^{ST} (ST interval), which is started every t_I^{CYC} time units (ST cycle) (Figure 4.2). In this interval, streams of ST class I have exclusive access to the link of the output port.*

Traffic from all other streams (including other ST classes) is gated (cut off) early enough by a guard band before the start of the next ST interval, such that the transmission of frames not belonging to streams of class I cannot overlap with I's ST interval.

Next, we present two resource-level analyses for IEEE 802.1Qbv: one for the streams of a ST class, and one for non-ST streams.

4.4.1.1 Resource-Level Analysis of Scheduled Traffic in IEEE 802.1Qbv

The frames of all ST streams i of a ST class I have exclusive access to an output port during class I's ST interval and, while they are waiting for their (first or next) ST interval to start, they are blocked. During their ST interval, they only experience same-priority blocking from frames of streams $j \in cl(i)$, e.g. when frames belonging to class $cl(i)$ arrive concurrently at an output port from different input ports (e.g. same-priority frames 1 to 3 in Figure 4.2). ST traffic streams do not experience any blocking from lower- or higher-priority traffic.

Thus, to compute the multiple-event processing time and the multiple-event scheduling horizon for a ST stream i under IEEE 802.1Qbv, we only have to consider two blocking terms: (1) same-priority blocking and (2) the blocking intervals during which no frames of ST stream i can be processed due to its class' transmission gate being closed or its class' guard band being active. In the following, we will refer to the latter blocking term as *closed-gate blocking*.[7]

[6] Note that IEEE 802.1Qbv does not require ST intervals to be periodic or of equal length (the gate schedule, however, will repeat eventually). We chose this periodic approach, because it seems to naturally fit many automotive use cases. Our analysis approach can be extended to also support other gate schedules, see Section 4.4.3.

[7] The closed-gate blocking will also include the blocking caused by an active guard band.

The same-priority blocking can be bounded as in IEEE 802.1Q.

Lemma 4 (Same-Priority Blocking of ST streams in IEEE 802.1Qbv)**.** *The same-priority blocking the q-th frame of a ST stream i that arrived at time a can experience can be bounded as for IEEE 802.1Q in Eq. (4.4).*

Proof. The same-priority blocking is independent of the number of ST intervals required to serve q frames of ST stream i. □

We model the blocking time between consecutive ST intervals of a ST class I as the time interval during which I's gate is closed plus the fraction of the ST interval's guard band which might be left unused because of non-preemption. Then, the amount of closed-gate blocking can be derived from the number of ST intervals required to serve the accumulated workload requested by ST class I. Abstractly, the number of required ST intervals can be derived by finding the worst-case combination among the arriving ST frames such that blocking (i.e. the number of ST intervals) is maximized, while each interval (except for the last one) is utilized such that no other ST frame can start transmission without finishing after the gate has closed. Since this is a discrete and combinatorially complex problem[8], we conservatively convert this into a continuous one by assuming that the ST intervals are only filled up to their guard band and attributing (in the analysis) any actual overlap into the guard band to the next ST interval.

We start by computing the minimum workload, which can be processed in a ST interval. Under the conservative assumption that any reasonable-sized ST interval should at least allow the transmission of one minimum-sized ST frame, i.e. a frame of size $\min_{i \in cl(i)}\{C_i^-\}$ (term (b) in Eq. (4.13)), we can state the following lemma.

Lemma 5. *The minimum workload, which can be processed in a ST interval of ST class $cl(i) = I \in \mathcal{S}$, is lower bounded by*

$$s^-_{cl(i)} = \max\Big\{ \underbrace{t^{ST}_{cl(i)} - \max_{i \in cl(i)}\left\{C_i^+\right\}}_{\text{(a)}}, \underbrace{\min_{i \in cl(i)}\left\{C_i^-\right\}}_{\text{(b)}} \Big\} \tag{4.13}$$

Proof. As term (b) is a (reasonable) assumption, we only need to proof term (a): To prevent the largest frame of ST class $cl(i)$ from transmitting beyond its ST interval boundaries, ST class $cl(i)$'s guard band must be of size $\max_{i \in cl(i)}\{C_i^+\}$. As long as there is backlogged workload, each ST interval is at least utilized for $t^{ST}_{cl(i)} - \max_{i \in cl(i)}\{C_i^+\}$. After this time interval the largest frame would not fit into the ST interval anymore. Hence, the minimum workload, which can be processed in a ST interval while there is backlogged workload can be lower bounded by term (a). Frames, whose transmission extends beyond the interval $t^{ST}_{cl(i)} - \max_{i \in cl(i)}\{C_i^+\}$ into the guard band are still allowed in an actual switch (see frame 2 in Figure 4.2), but their overlap into the guard band is

[8]Note that this is not a bin packing problem, as bin packing would try to find the minimum number of ST intervals (bins), i.e. the minimum blocking.

not considered to be part of the minimum workload in term (a) in the analysis. Instead, the overlap is (in our analysis) shifted to the next ST interval. This is a conservative overapproximation of the discrete problem, i.e. it maximizes the number of ST intervals required to process a given workload, as the analysis always processes workload of exactly $t^{ST}_{cl(i)} - \max_{i \in cl(i)} \left\{C_i^+\right\}$ in all but the last ST interval, while the actual system processes workload from the range $(t^{ST}_{cl(i)} - \max_{i \in cl(i)} \left\{C_i^+\right\}, t^{ST}_{cl(i)}]$ (depending on the actual frame sizes) in all but the last ST interval.

As terms (a) and (b) are conservative bounds, we can take their maximum. □

With Lemma 5, we can upper bound the closed-gate blocking.

Lemma 6 (Closed-Gate Blocking in IEEE 802.1Qbv)**.** *The closed-gate blocking of a ST stream i of ST class $cl(i)$ with an accumulated workload of Δw is the maximum time this workload can be kept from being processed due to its transmission gate being closed. It can be upper bounded by:*

$$I_i^{CGB}(\Delta w) = \underbrace{t^{CYC}_{cl(i)} - t^{ST}_{cl(i)} + \max_{i \in cl(i)} \{C_i^+\}}_{\text{(a)}} + \underbrace{\left(\left\lceil \frac{\Delta w}{s^-_{cl(i)}} \right\rceil - 1 \right) \left(t^{CYC}_{cl(i)} - s^-_{cl(i)} \right)}_{\text{(b)}} \tag{4.14}$$

Proof. Term (a) gives an upper bound on the longest time the queued workload has to wait for service, i.e. due to its transmission gate being closed or its guard band being active, for the first time. In the worst-case the longest frame of ST class $cl(i)$ (i.e. the frame of the length of ST class $cl(i)$'s guard band) arrives just after ST class $cl(i)$'s guard band started. Then it takes $t^{CYC}_{cl(i)} - t^{ST}_{cl(i)}$ time to reach the next (i.e. first usable) ST interval (see Figure 4.2).

Term (b): The maximum number of intervals during which the queued workload has to wait for service after the first can be upper bounded by dividing Δw by $s^-_{cl(i)}$, i.e. a lower bound on the minimum amount of service that is processed during each ST interval, rounding the result up, and reducing this number by 1 (the first interval has already been considered by term (a)). An upper bound on the blocking time per interval can be computed by subtracting the minimum amount of service that is processed during each ST interval from the ST cycle length: $t^{CYC}_{cl(i)} - s^-_{cl(i)}$. The overall blocking time from all intervals during which the queued workload has to wait for service after the first can be upper bounded by multiplication. □

With Eqs. (4.4) and (4.14), the multiple-event queueing delay for ST streams can be bounded.

Theorem 21. *The multiple-event queueing delay in IEEE 802.1Qbv for the q-th frame arrival of ST stream i that arrived at time a is upper bounded by*

$$Q_{i,a}(q) = I_i^{SPB}(q,a) + I_i^{CGB}\left(I_i^{SPB}(q,a) + C_i^+\right) \tag{4.15}$$

Proof. The multiple-event queueing delay must take all blocking effects into account. From Lemmata 2 and 6 the theorem follows. As $I_i^{CGB}(\Delta w)$ requires the accumulated workload requested by ST class $cl(i)$ as its argument and $I_i^{SPB}(q,a)$ only considers $(q-1)C_i^+$, we must add C_i^+. □

With the multiple-event queueing delay, the multiple-event processing time for ST streams can be bounded.

Theorem 22. *The multiple-event processing time for the q-th frame arrival of a ST stream i in IEEE 802.1Qbv is upper bounded by*

$$B_i^+(q) = Q_i(q) + C_i^+ \tag{4.16}$$

where $Q_i(q)$ is an upper bound on the multiple-event queueing delay for the q-th frame arrival of a ST stream i in IEEE 802.1Qbv:

$$Q_i(q) = I_i^{SPB}(q, Q_i(q)) + I_i^{CGB}\left(I_i^{SPB}(q, Q_i(q)) + C_i^+\right) \tag{4.17}$$

Proof. Follows the proofs of Theorem 12 with Lemmata 2 and 6. □

With the help of the same-priority and closed-gate blocking terms, also, the multiple-event scheduling horizon for ST streams can be derived.

Theorem 23. *The multiple-event scheduling horizon for the q-th frame arrival of a ST stream i in IEEE 802.1Qbv is upper bounded by*

$$S_i(q) = \tilde{I}_i^{SPB}(q, S_i(q)) + I_i^{CGB}\left(\tilde{I}_i^{SPB}(q, S_i(q)) + C_i^+\right) \tag{4.18}$$

with

$$\tilde{I}_i^{SPB}(q, \Delta t) = qC_i^+ + \sum_{j \in fifo(i)} \eta_j^+(\Delta t) C_j^+ \tag{4.19}$$

Proof. The scheduling horizon of the q-th frame arrival of ST stream i ends, when all q frames of ST stream i have been transmitted and all frames of interfering ST streams that arrived within the scheduling horizon of the q-th frame arrival of ST stream i have been transmitted as well. Only then an additional hypothetical activation of infinitesimal small execution time could be processed. The amount of frames from interfering traffic streams can be bounded by Lemmata 2, and 6. However, similar to Theorem 20, we have to adjust the blocking terms from Eq. (4.4) with Eqs. (4.19) to address the fact that the scheduling horizon is defined over a right half-open interval, i.e. we use $\eta_j^+(\Delta t)$ instead of $\eta_j^{+]}(\Delta t)$. Additionally, we need to consider all q frame arrivals when computing the multiple-event scheduling horizon for the q-th frame arrival of ST stream i. This is also addressed by Eq. (4.19). □

The worst-case frame forwarding times and maximum buffer size requirements for ST streams in IEEE 802.1Qbv can be computed as described in Section 4.3.2 and Section 4.3.4.

4.4.1.2 Resource-Level Analysis of Non-Scheduled Traffic in IEEE 802.1Qbv

Frames of non-ST traffic streams $i \in \bigcup_{I \notin \mathcal{S}} I$ interfere with all other lower- and higher-priority streams of other non-ST classes. They are not part of any ST interval (all non-ST gates are open while all ST gates are closed in our model) and receive the leftover bandwidth. Hence, we can model non-ST traffic like IEEE 802.1Q and consider ST intervals from all classes $J \in \mathcal{S}$ as additional periodic blocking terms, when deriving the worst-case queueing delay.

Specifically, lower-, same-, and higher-priority blocking from non-ST traffic streams can be modeled as in Lemmata 1, 2, and 3, while bearing in mind that only interfering traffic from non-ST traffic must be considered for lower- and higher-priority blocking.

Lemma 7 (Lower-Priority Blocking of Non-ST Streams in IEEE 802.1Qbv)**.** *The lower-priority blocking a non-ST stream i can experience in IEEE 802.1Qbv is bounded by*

$$I_i^{LPB} = \max_{j \in \{k \mid k \in lp(i) \wedge cl(k) \notin \mathcal{S}\}} \left\{ C_j^+ \right\} \tag{4.20}$$

where $lp(i)$ is the set of all traffic streams with a priority lower than that of stream i.

Proof. Follows Lemma 1. □

The same-priority blocking for non-ST streams can be bounded as in IEEE 802.1Q.

Lemma 8 (Same-Priority Blocking of Non-ST Streams in IEEE 802.1Qbv)**.** *The same-priority blocking the q-th frame of a non-ST stream i that arrived at time a can be bounded as for IEEE 802.1Q in Eq. (4.4).*

Proof. Follows Lemma 2. □

Lemma 9 (Higher-Priority Blocking of Non-ST Streams in IEEE 802.1Qbv)**.** *The higher-priority blocking a non-ST stream i can experience in any time interval Δt in IEEE 802.1Qbv is bounded by*

$$I_i^{HPB}(\Delta t) = \sum_{j \in \{k \mid k \in hp(i) \wedge cl(k) \notin \mathcal{S}\}} \eta_j^{+]}(\Delta t) C_j^+ \tag{4.21}$$

where $hp(i)$ is the set of all traffic streams with a priority higher than that of stream i.

Proof. Follows Lemma 3. □

The additional periodic blocking by the ST intervals of ST classes can be bounded as follows. First, we bound the maximum blocking a single ST interval can cause.

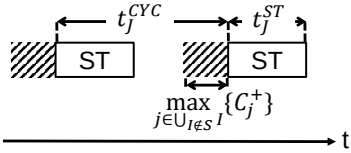


Figure 4.3: Illustration of the terms contributing to the maximum blocking time of the IEEE 802.1Qbv analysis for non-ST traffic streams

Lemma 10. *The maximum blocking caused by a single ST interval t_J^{ST} from ST class $J \in \mathcal{S}$ on non-ST classes in IEEE 802.1Qbv is bounded by*

$$\tilde{t}_J^{ST} = \max_{i \in \bigcup\limits_{I \notin \mathcal{S}} I} \left\{ C_i^+ \right\} + t_J^{ST} \tag{4.22}$$

Proof. The first term models that, to guard the ST interval of ST class J, its guard band must be large enough such that even the longest non-ST frame cannot overlap into J's ST interval (Figure 4.3). The second term is the ST interval length of ST class J. □

Now, we can formulate the blocking by ST classes.

Lemma 11 (Blocking by ST Classes in IEEE 802.1Qbv)**.** *In any time interval Δt, the maximum interference the frames of a non-ST stream i can experience by a ST class J is bounded by*

$$I_i^{STB}(\Delta t) = \sum_{J \in \mathcal{S}} \left\lceil \frac{\Delta t}{t_J^{CYC}} \right\rceil \tilde{t}_J^{ST} \tag{4.23}$$

Proof. We conservatively assume that, in the worst-case, the blocking by different ST intervals does not overlap. In any time interval Δt, the maximum number of times a ST class J can block frames of a non-ST stream i with its ST interval can be derived by diving Δt by t_J^{CYC}. The actual blocking time is bounded by multiplication with the maximum blocking caused by a single ST interval of ST class J.[9] □

With Eqs. (4.20), (4.4), (4.21), and (4.23) the multiple-event queueing delay for non-ST streams can be bounded.

Theorem 24. *The multiple-event queueing delay in IEEE 802.1Qbv for the q-th frame arrival of non-ST stream i that arrived at time a is upper bounded by*

$$Q_{i,a}(q) = I_i^{LPB} + I_i^{SPB}(q, a) + I_i^{HPB}(Q_{i,a}(q)) + I_i^{STB}(Q_{i,a}(q)) \tag{4.24}$$

[9] Note that IEEE 802.1Qbv explicitly does not distinguish between ST and non-ST intervals. Hence, the guard band mechanism also applies to ST intervals, i.e. ST frames cannot overlap into non-ST intervals.

Proof. The multiple-event queueing delay must take all blocking effects into account. From Lemmata 7, 2, 9 and 11 the theorem follows. □

With the multiple-event queueing delay, the multiple-event processing time for non-ST streams can be bounded.

Theorem 25. *The multiple-event processing time for the q-th frame arrival of a non-ST stream i in IEEE 802.1Qbv is upper bounded by*

$$B_i^+(q) = Q_i(q) + C_i^+ \tag{4.25}$$

where $Q_i(q)$ is an upper bound on the multiple-event queueing delay for the q-th frame arrival of a non-ST stream i in IEEE 802.1Qbv:

$$Q_i(q) = I_i^{LPB} + I_i^{SPB}(q, Q_i(q)) + I_i^{HPB}(Q_i(q)) + I_i^{STB}(Q_i(q)) \tag{4.26}$$

Proof. Follows the proofs of Theorem 9, and Theorem 12 with Lemmata 7, 2, 9, and 11. □

With the help of the lower-, same-, and higher priority blocking terms as well as the blocking by ST classes, also, the multiple-event scheduling horizon for non-ST streams can be derived.

Theorem 26. *The multiple-event scheduling horizon for the q-th frame arrival of a non-ST stream i in IEEE 802.1Qbv is upper bounded by*

$$S_i(q) = I_i^{LPB} + \tilde{I}_i^{SPB}(q, S_i(q)) + \tilde{I}_i^{HPB}(S_i(q)) + I_i^{STB}(S_i(q)) \tag{4.27}$$

with

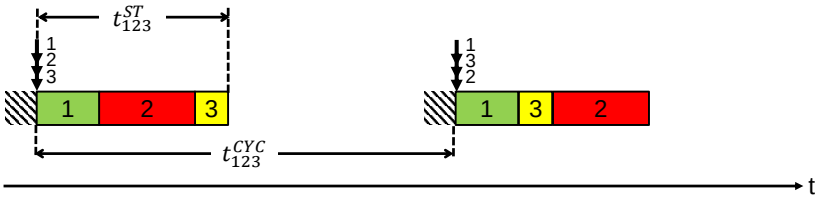
$$\tilde{I}_i^{SPB}(q, \Delta t) = qC_i^+ + \sum_{j \in fifo(i)} \eta_j^+(\Delta t) C_j^+ \tag{4.28}$$

and

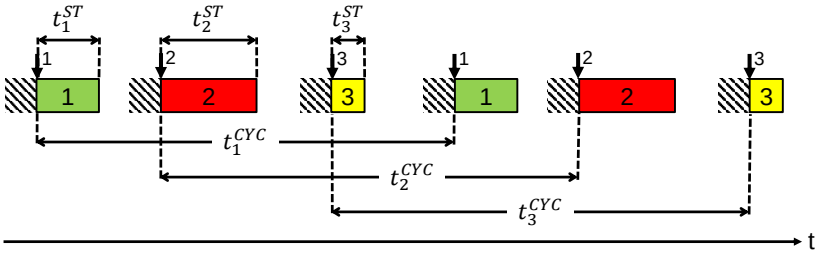
$$\tilde{I}_i^{HPB}(\Delta t) = \sum_{j \in \{k \mid k \in hp(i) \wedge cl(k) \notin \mathcal{S}\}} \eta_j^+(\Delta t) C_j^+ \tag{4.29}$$

Proof. The scheduling horizon of the q-th frame arrival of traffic stream i ends, when all q frames of stream i have been transmitted and all frames of interfering traffic streams that arrived within the scheduling horizon of the q-th frame arrival of traffic stream i have been transmitted as well. Only then an additional hypothetical activation of infinitesimal small execution time could be processed. The amount of frames from interfering traffic streams can be bounded by Lemmata 7, 2, and 9. However, similar to Theorem 20, we have to adjust the blocking terms from Eqs. (4.4) and (4.21) with Eqs. (4.28) and (4.29) (respectively) to address the fact that the scheduling horizon is defined over a right half-open interval, i.e. we use $\eta_j^+(\Delta t)$ instead of $\eta_j^{+]}(\Delta t)$. Additionally, we need to consider all q frame arrivals when computing the multiple-event scheduling horizon for the q-th frame arrival of traffic stream i. This is also addressed by Eq. (4.28). □

The worst-case frame forwarding times and maximum buffer size requirements for non-ST streams in IEEE 802.1Qbv can be computed as described in Section 4.3.2 and Section 4.3.4.



(a) Integration mode 1



(b) Integration mode 2

Figure 4.4: Examples of (a) integration mode 1 and (b) integration mode 2

4.4.2 Support for Integration Modes 1 and 2

The formulae in the previous sections support both integration mode 1 and 2. Integration mode 1 is supported by explicitly considering same-priority blocking even for ST classes in the multiple-event queueing delay (Eq. (4.15)) and the multiple-event scheduling horizon (Eq. (4.18)). Figure 4.4a illustrates integration mode 1. Here, ST frames 1, 2, and 3 are all mapped to ST class 123. Note that the there is no constraint on the order in which ST frames are mapped into the ST intervals (unless enforced by tightly timed frame releases). The second ST interval shows such a different frame order.

Integration mode 2 is supported by simply assigning only a single ST stream to a ST class. This can be done, because for ST traffic, the closed-gate blocking in Eq. (4.14) only depends on parameters of the ST class under analysis. Interference from other ST classes as well as from non-ST classes is covered by the closed-gate blocking without the need for explicit modeling. For non-ST traffic, the interference from ST classes is considered by summing over all ST classes in Eq. (4.23) independent of their priority. Note that in the analysis implementation, traffic classes are distinguished by their priority. This means that, in integration mode 2, each ST class must be mapped to a dedicated analysis priority. Depending on the number of ST classes, there may be the need to use more analysis priorities than Ethernet priorities. This does not impose a problem for non-overlapping ST intervals of integration mode 2, however, since, as outline above, the analysis priority of ST classes does not have an impact on the analysis result. Figure 4.4b illustrates integration mode 2. Here, ST frames

1, 2, and 3 are mapped to individual ST classes 1, 2, and 3, each with its own guard band.

4.4.3 Support for Arbitrary-Placed ST Intervals of Non-Uniform Length

The outlined analysis method supports multiple ST streams, whose ST intervals have a fixed, uniform length and are scheduled periodically. IEEE 802.1Qbv's gate control list, however, is not limited to periodically-placed, uniform length ST intervals and also allows to configure arbitrary-placed ST intervals of non-uniform length (i.e. a ST stream can have multiple ST intervals at arbitrary locations each of potentially different length). In this section, we sketch how our analysis can be extended to support such general gate schedules. We start by stating the constraints of IEEE 802.1Qbv and then outline the extensions for both scheduled traffic and non-scheduled traffic.

The key observation is the schedule imposed by the gate control list. The gate control list is a fixed-length, repeating schedule that defines at which times the gates of ST classes open and close. This implies that both the times at which the ST intervals of each ST class start and end (potentially multiple times within the gate schedule) are fixed and the lengths of each of these individual ST intervals is also fixed (and may potentially be different for each ST interval for the same ST class in the gate schedule). The gate schedule repeats eventually, yielding hyperperiodic behavior. This can be exploited by the analysis.

4.4.3.1 Scheduled Traffic

For ST streams, we have to adjust Eq. (4.14) for the closed-gate blocking to consider the gate schedule for the ST class under analysis. As ST intervals may be placed at arbitrary points in the gate schedule and may be of different length, we must determine at which time, i.e. offset in the gate schedule, to start the critical instant scenario for the analysis. Concretely, we have to pick, for a given accumulated workload Δw, the offset in the gate schedule that leads to the longest closed-gate blocking.

4.4.3.2 Non-Scheduled Traffic

For non-ST streams, we have to adjust Eq. (4.23) for blocking by ST classes to consider the gate schedules of all ST classes. Akin to scheduled traffic, we have to pick, for a given time interval Δt, an offset in the gate schedule that leads to the maximum blocking by ST intervals to construct the critical instant scenario.

4.4.4 Design Considerations

ST streams are subject to same-priority blocking and closed-gate blocking (cf. Eq. (4.15)). While we cannot eliminate the same-priority blocking, we can try to eliminate the closed-gate blocking. Typically, and this is the initial motivation

behind scheduled traffic in IEEE 802.1Qbv, one would engineer IEEE 802.1Qbv networks such that (1) all ST intervals are sufficiently large and (2) all gate control lists of all ports along a path are synchronized.

The first requirement implies that the ST intervals should be large enough to process all frames that arrive during one ST cycle. This prevents the multiple-event scheduling horizon of ST streams to be spread over more than one ST interval, which would imply (additional) closed-gate blocking, i.e. it eliminates term (b) in Eq. (4.14). In order to meet this requirement under worst-case conditions, we must make sure that, for all ST classes $I \in \mathcal{S}$, the maximum multiple-event processing time for each ST stream $i \in I$ still fits into a single ST interval while also taking the maximum guard band length into account (cf. Eq. (4.13)), i.e.

$$\forall i \in \bigcup_{I \in \mathcal{S}} I : B_i^+(\hat{q}_i) \leq s^-_{cl(i)} \tag{4.30}$$

The second requirement implies that all ST intervals of a ST class I should be aligned such that I's frames do not need to wait for their ST interval to start. In Lemma 6 we conservatively assumed that, in the worst-case, the first frame just missed its ST interval and has to wait for the next one. This worst-case scenario can occur, if the gate control lists throughout the network are not synchronized, but can be prevented by careful design and strict enforcement of time synchronization (e.g. by IEEE 802.1AS). This eliminates term (a) in Eq. (4.14). We will additionally consider a setup, where all gate control lists are assumed to be (perfectly) synchronized, in our evaluation in Chapter 5. Note that this, however, has synchronization implications on all switches and nodes in the network and also limits topology and traffic flow choices (e.g. rejoining paths might become a complex task).

4.5 Analysis of IEEE 802.1Qch

As discussed in Section 2.2.3.3, IEEE 802.1Qch [80] uses cyclic frame forwarding on top of SPNP scheduling to easily bound the residence time of frames in switches. This cyclic forwarding is also called peristaltic forwarding or peristaltic shaping (PS). The main principle of IEEE 802.1Qch is that time is split in alternating (peristaltic) intervals of fixed equal length, referred to as even and odd. All frames that have been received in an even interval, will be scheduled to be transmitted in the next odd one and vice versa.

If configured correctly, the latency of a frame can then be computed independently of interfering traffic as a function of the number of hops and the cyclic interval length. Transient overload, i.e. if the workload received in an interval would exceed the next interval's length, must be handled appropriately. IEEE 802.1Qch [80] proposes dropping or priority reduction. A different approach is to continue processing the overload [165], but this would put the latency guarantees at risk.

Our CPA-based analysis derives timing guarantees (even) for the latter case and can be configured to generate a warning when the received workload

will exceed its PS interval. Our analysis also assumes that, under worst-case conditions, the cyclic intervals are not synchronized.

4.5.1 Resource-Level Analysis of IEEE 802.1Qch

As with IEEE 802.1Q and IEEE 802.1Qbv, in order to integrate the performance analysis of IEEE 802.1Qch into the resource-level analysis framework outlined in Section 3.3.1, we need to define and compute the multiple-event scheduling horizon and the multiple-event processing time as well as as the multiple-event queueing delay.

In the following two sections, we present two resource-level analyses for IEEE 802.1Qch: one for streams of a non-PS class, and one for PS streams. However, first, in order to formalize these resource-level analyses, we introduce a set of definitions.

We assume that there are PS and non-PS traffic classes, streams and frames.

Definition 28 (PS Classes, PS Streams, and PS Frames)**.** *Let $\mathcal{C}$ be the set of peristaltic (PS) traffic classes, i.e. the set of all Ethernet classes which are shaped by a peristaltic shaper based on their arrival interval. We call traffic classes $I \in \mathcal{C}$ PS classes and, correspondingly, their traffic streams PS streams and frames of these streams PS frames.*

Likewise non-PS classes, non-PS streams, and non-PS frames are defined for traffic classes $J \notin \mathcal{C}$.

Furthermore, we assume that each PS class can have its own cyclically repeating PS interval.

Definition 29 (PS Interval Length)**.** *Let $t_I^{PS} > 0$ be the peristaltic (PS) interval length of a given PS class $I \in \mathcal{C}$.*

In IEEE 802.1Qch, after a frame of PS stream i arrives at an output port, it must wait for its current PS interval to end until it can take part in the arbitration process of the port's scheduler, regardless of whether the port is busy processing other frames during this waiting time or not, i.e. IEEE 802.1Qch is not work-conserving. As we will see later, the length of the scheduling horizon depends on how the periodic PS interval pattern is aligned relative to the start of the scheduling horizon.

Definition 30 (PS Offset)**.** *For a given PS class I, the time interval from the start of the scheduling horizon until the first full PS interval within the scheduling horizon is called the initial PS offset $\phi_I^{PS} \in (0, t_I^{PS}]$ of PS class I.*

4.5.1.1 Resource-Level Analysis of Non-Peristaltic Traffic in IEEE 802.1Qch

The underlying transmission selection mechanism in IEEE 802.1Qch is IEEE 802.1Q's SPNP mechanism. Hence, we need to consider lower-, same-, and higher-priority blocking terms when deriving bounds for the multiple-event processing time, the multiple-event queueing delay, and the multiple-event scheduling horizon. Additionally, we have to take any effects of the peristaltic

forwarding of interfering PS traffic, i.e. the PS intervals, into account when required.

The lower-priority blocking can be bounded as in IEEE 802.1Q.

Lemma 12 (Lower-Priority Blocking of non-PS traffic streams in IEEE 802.1Qch)**.** *The lower-priority blocking a non-PS traffic stream i can experience in IEEE 802.1Qch is bounded as for IEEE 802.1Q in Eq. (4.3).*

Proof. The lower-priority blocking constitutes a single non-preemptable frame from a PS or non-PS traffic stream with the largest frames with a priority lower than that of non-PS stream i (cf. Lemma 1). After this frame has completed its transmission, frames of the higher-priority traffic streams (including non-PS traffic stream i) are processed. Consequently, it is independent of any PS intervals or PS offsets. □

Also, the same-priority blocking can be bounded as in IEEE 802.1Q.

Lemma 13 (Same-Priority Blocking of non-PS traffic streams in IEEE 802.1Qch)**.** *The same-priority blocking the q-th frame of a non-PS traffic stream i that arrived at time a can experience can be bounded as for IEEE 802.1Q in Eq. (4.4).*

Proof. The same-priority blocking is caused by FIFO scheduling in class $cl(i)$'s output queues. Consequently, the blocking of the q-th frame of a non-PS traffic stream i only depends on its $q-1$ predecessors and on the frames from interfering same-priority traffic classes that arrived until time a. It is independent of any PS intervals or PS offsets. □

Higher-priority blocking, however, requires special attention and we need to distinguish two cases: (1) the higher-priority blocking is by frames of non-PS streams j, i.e. $j \in hp(i) \wedge cl(j) \notin \mathcal{C}$, and (2) the higher-priority blocking is by frames of PS streams j, i.e. $j \in hp(i) \wedge cl(j) \in \mathcal{C}$.

In the first case, higher-priority blocking on frames of stream i can be computed as in IEEE 802.1Q.

Lemma 14 (Higher-Priority Blocking of non-PS traffic streams from non-PS traffic streams in IEEE 802.1Qch)**.** *The higher-priority blocking a non-PS traffic stream i can experience from non-PS traffic streams in any time interval Δt in IEEE 802.1Qch is bounded by*

$$I_i^{HPB,non-PS}(\Delta t) = \sum_{j \in \{k | k \in hp(i) \wedge cl(k) \notin \mathcal{C}\}} \eta_j^{+]}(\Delta t)\, C_j^+ \tag{4.31}$$

Proof. Higher-priority non-PS traffic streams are not affected by cyclic frame forwarding. Hence, their interference can be bounded as in IEEE 802.1Q (see Lemma 3), including the critical instant scenario. □

In the second case, we have to consider that frames of the streams of a higher-priority PS class $J \in \mathcal{C}$ might not be presented to the scheduler immediately.

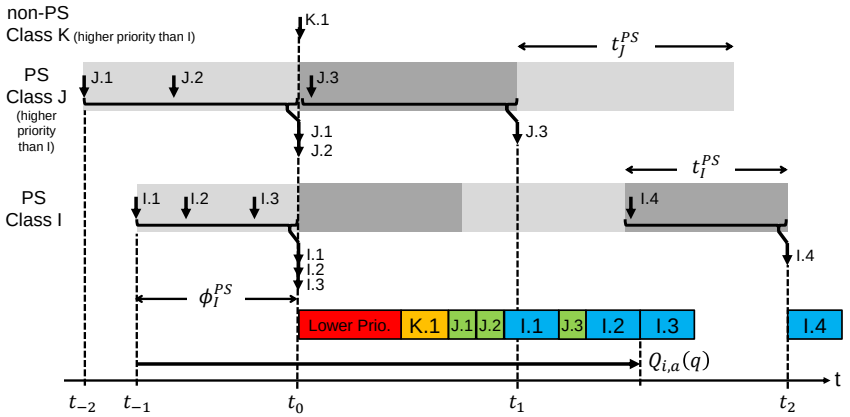


Figure 4.5: IEEE 802.1Qch: Frames of PS class I experience interference from higher priority classes K (non-PS) and J (PS) and one lower-priority frame. It takes more than one PS interval to transmit the first three frames of PS stream I.

Lemma 15 (Higher-Priority Blocking of non-PS traffic streams from PS traffic streams in IEEE 802.1Qch)**.** *The higher-priority blocking a non-PS traffic stream i can experience from PS traffic streams in any time interval Δt in IEEE 802.1Qch is bounded by*

$$I_i^{HPB,PS}(\Delta t) = \sum_{j \in \{k | k \in hp(i) \wedge cl(k) \in \mathcal{C}\}} \eta_j^{+]} \left(\left\lceil \frac{\Delta t}{t_{cl(j)}^{PS}} \right\rceil t_{cl(j)}^{PS} \right) C_j^+ \tag{4.32}$$

Proof. In contrast to the critical instant scenario for higher-priority interference from non-PS traffic streams, we have to consider a different critical instant scenario to upper bound the amount of interference from PS traffic streams: In the worst-case, frames of higher-priority PS traffic streams j have been delayed for the length of their PS interval $t_{cl(j)}^{PS}$, before they can interfere with the frames of stream i, i.e. the critical instant scenario for interfering PS traffic streams is such that the end of the first PS interval of each interfering higher-priority PS class coincides with the arrival of the first of the q frames of non-PS traffic stream i and that the frames of the traffic streams of these interfering PS classes have been arriving as fast as possible since the beginning of their first PS interval.

For any time interval Δt, the maximum number of PS intervals of PS class $cl(j)$, which end in this Δt, can be computed by $\lceil \Delta t / t_{cl(j)}^{PS} \rceil$. Multiplying this number by the PS interval length of PS class $cl(j)$ yields the time interval during which the interference from PS streams j has been accumulated, i.e. $\lceil \Delta t / t_{cl(j)}^{PS} \rceil t_{cl(j)}^{PS}$. We can use this time interval to compute the higher-priority blocking of each stream j on stream i as in Eq. (4.32). As in Lemma 3, we use the closed-interval upper arrival function to consider frames of higher-priority PS traffic streams arrive just at the end of Δt as additional interference. □

Lemma 15 is illustrated in Figure 4.5 for PS class J, where, for example, frames $J.1$ and $J.2$ are delayed during a PS interval until they are released concurrently at the interval's end.

We can combine both higher-priority blocking terms.

Corollary 1 (Higher-Priority Blocking of non-PS traffic streams in IEEE 802.1Qch)**.** *The higher-priority blocking a non-PS traffic stream i can experience in any time interval Δt in IEEE 802.1Qch is bounded by*

$$I_i^{HPB}(\Delta t) = I_i^{HPB,non-PS}(\Delta t) + I_i^{HPB,PS}(\Delta t) \tag{4.33}$$

Proof. Follows from Lemma 14 and Lemma 15, as $\{k|k \in hp(i) \wedge cl(k) \notin \mathcal{C}\} \cup \{k|k \in hp(i) \wedge cl(k) \in \mathcal{C}\} = hp(i)$ and $\{k|k \in hp(i) \wedge cl(k) \notin \mathcal{C}\} \cap \{k|k \in hp(i) \wedge cl(k) \in \mathcal{C}\} = \emptyset$. □

With Lemma 12, Lemma 13, and Corollary 1 the multiple-event queueing delay for non-PS traffic streams can be bounded.

Theorem 27. *The multiple-event queueing delay in IEEE 802.1Qch for the q-th frame arrival of non-PS traffic stream i that arrived at time a is upper bounded by*

$$Q_{i,a}(q) = I_i^{LPB} + I_i^{SPB}(q,a) + I_i^{HPB}(Q_{i,a}(q)) \tag{4.34}$$

Proof. The multiple-event queueing delay must take all blocking effects into account. From Lemma 12, Lemma 13, and Corollary 1 the theorem follows. □

With the multiple-event queueing delay, the multiple-event processing time for non-PS traffic streams can be bounded.

Theorem 28. *The multiple-event processing time for the q-th frame arrival of a non-PS traffic stream i in IEEE 802.1Qch is upper bounded by*

$$B_i^+(q) = Q_i(q) + C_i^+ \tag{4.35}$$

where $Q_i(q)$ is an upper bound on the multiple-event queueing delay for the q-th frame arrival of an non-PS stream i in IEEE 802.1Qch:

$$Q_i(q) = I_i^{LPB} + I_i^{SPB}(q, Q_i(q)) + I_i^{HPB}(Q_i(q)) \tag{4.36}$$

Proof. Follows the proofs of Theorems 9 and 12 with Lemmata 12, 13, and Corollary 1. □

With the help of the lower-, same-, and higher priority blocking terms, also, the multiple-event scheduling horizon for non-PS streams can be derived.

Theorem 29. *The multiple-event scheduling horizon for the q-th frame arrival of a non-PS traffic stream i in IEEE 802.1Qch is upper bounded by*

$$S_i(q) = I_i^{LPB} + \tilde{I}_i^{SPB}(q, S_i(q)) + \tilde{I}_i^{HPB}(S_i(q)) \tag{4.37}$$

with

$$\tilde{I}_i^{SPB}(q, \Delta t) = qC_i^+ + \sum_{j \in fifo(i)} \eta_j^+(\Delta t) C_j^+ \tag{4.38}$$

and

$$\tilde{I}_i^{HPB}(\Delta t) = \sum_{j \in \{k | k \in hp(i) \wedge cl(k) \notin \mathcal{C}\}} \eta_j^+(\Delta t)\, C_j^+ + \sum_{j \in \{k | k \in hp(i) \wedge cl(k) \in \mathcal{C}\}} \eta_j^+ \left(\left\lceil \frac{\Delta t}{t_{cl(j)}^{PS}} \right\rceil t_{cl(j)}^{PS} \right) C_j^+ \tag{4.39}$$

Proof. The scheduling horizon of the q-th frame arrival of non-PS traffic stream i ends, when all q frames of stream i have been transmitted and all frames of interfering traffic streams that arrived within the scheduling horizon of the q-th frame arrival of non-PS traffic stream i have been transmitted as well. Only then an additional hypothetical activation of infinitesimal small execution time could be processed. The amount of frames from interfering traffic streams can be bounded by Lemmata 12, 13, and Corollary 1. However, similar to Theorem 20, we have to adjust the blocking terms from Eqs. (4.4) and (4.33) with Eqs. (4.38) and (4.39) to address the fact that the scheduling horizon is defined over a right half-open interval, i.e. we use $\eta_j^+(\Delta t)$ instead of $\eta_j^{+]}(\Delta t)$. Additionally, we need to consider all q frame arrivals when computing the multiple-event scheduling horizon for the q-th frame arrival of traffic stream i. This is also addressed by Eq. (4.38). □

The worst-case frame forwarding times and maximum buffer size requirements for non-PS traffic streams in IEEE 802.1Qch can be computed as described in Section 4.3.2 and Section 4.3.4.

4.5.1.2 Resource-Level Analysis of Peristaltic Traffic in IEEE 802.1Qch

The analysis of a PS traffic stream i is similar to the analysis of non-PS traffic streams, except for the initial offset $\phi_{cl(i)}^{PS}$ of the PS traffic stream i under analysis, which must be considered when required.

The lower-priority blocking can be bounded as in IEEE 802.1Q.

Lemma 16 (Lower-Priority Blocking of PS traffic streams in IEEE 802.1Qch)**.** *The lower-priority blocking a PS traffic stream i can experience in IEEE 802.1Qch is bounded as for IEEE 802.1Q in Eq. (4.3).*

Proof. The lower-priority blocking constitutes a single non-preemptable frame from a PS or non-PS traffic stream with the largest frames with a priority lower than that of PS stream i (cf. Lemma 1), which starts interfering with frames of PS traffic stream i just before $\phi_{cl(i)}^{PS}$ ends, i.e. just before the first of the q frames of PS traffic stream i can be considered by the transmission selection mechanism. However, the amount of interference is independent of the actual value of $\phi_{cl(i)}^{PS}$. □

Also, the same-priority blocking can be bounded as in IEEE 802.1Q.

Lemma 17 (Same-Priority Blocking of PS traffic streams in IEEE 802.1Qch)**.** *The same-priority blocking the q-th frame of a PS traffic stream i that arrived at time a can experience can be bounded as for IEEE 802.1Q in Eq. (4.4).*

Proof. Follows proof of Lemma 13. □

Note that the same-priority blocking does not contain the initial PS offset. We will discuss the blocking introduced by this term later.

The higher-priority blocking can be bounded as for non-PS traffic streams.

Lemma 18 (Higher-Priority Blocking of PS traffic streams in IEEE 802.1Qch)**.** *The higher-priority blocking a non-PS traffic stream i can experience in any time interval Δt in IEEE 802.1Qch can be bounded by Eqs. (4.31) and (4.32) and, consequently, Eq. (4.33).*

Proof. Follows from proofs of Lemma 14, Lemma 15, and Corollary 1. For Eq. (4.32), however, the critical instant scenario is slightly different (without impacting Eqs. (4.31) and (4.32)): In the worst-case, frames of higher-priority PS traffic streams j have been delayed for the length of their PS interval $t^{PS}_{cl(j)}$, before they can interfere with the frames of stream i, i.e. the critical instant scenario for interfering PS traffic streams is such that the end of the first PS interval of each interfering higher-priority PS class coincides with the time when the first of the q frames of PS traffic stream i could be considered by the transmission selection mechanism, i.e. just before PS traffic stream i's PS offset ends at $\phi^{PS}_{cl(i)}$. As before, in this critical instant scenario the frames of the traffic streams of the interfering higher-priority PS classes are assumed to arrive as fast as possible from beginning of their first PS interval. □

In Figure 4.5, for example, PS class J is aligned such that frames $J.1$ and $J.2$ from the first PS interval of J start interfering with PS class I at t_0, i.e. just when ϕ^{PS}_I ended.

With Lemma 16, Lemma 17, and Lemma 18 the multiple-event queueing delay for a PS traffic stream i can be bounded by additionally taking its initial offset $\phi^{PS}_{cl(i)}$ into account.

Theorem 30. *The multiple-event queueing delay in IEEE 802.1Qch for the q-th frame arrival of a PS traffic stream i that arrived at time a is upper bounded by*

$$Q_{i,a}(q) = \tilde{Q}_{i,a}(q) + \phi^{PS}_{cl(i)} \tag{4.40}$$

with

$$\tilde{Q}_{i,a}(q) = I^{LPB}_i + I^{SPB}_i(q,a) + I^{HPB}_i(\tilde{Q}_{i,a}(q)) \tag{4.41}$$

Proof. The multiple-event queueing delay must take all blocking effects into account. The terms modeling the blocking from other frames are taken from Lemma 16, Lemma 17, and Lemma 18. Additionally, the first time frames of PS traffic class $cl(i)$ could be considered by the transmission selection mechanism

is after their PS offset at $\phi^{PS}_{cl(i)}$. Note that $\phi^{PS}_{cl(i)}$ only affects frames of streams $i \in cl(i)$, i.e. frames of traffic streams of other classes are not affected (blocked) during $\phi^{PS}_{cl(i)}$. Consequently, $\phi^{PS}_{cl(i)}$ is added after the fixed point iteration (over $\tilde{Q}_{i,a}(q)$) that collects interfering traffic. □

With the multiple-event queueing delay, the multiple-event processing time for PS traffic streams can be bounded.

Theorem 31. *The multiple-event processing time for the q-th frame arrival of a PS traffic stream i in IEEE 802.1Qch is upper bounded by*

$$B^+_i(q) = Q_i(q) + C^+_i \tag{4.42}$$

where $Q_i(q)$ is an upper bound on the multiple-event queueing delay for the q-th frame arrival of a PS traffic stream i in IEEE 802.1Qch:

$$Q_i(q) = \tilde{Q}_i(q) + \phi^{PS}_{cl(i)} \tag{4.43}$$

with

$$\tilde{Q}_i(q) = I^{LPB}_i + I^{SPB}_i\left(q, \underbrace{\left(\left\lceil \frac{\tilde{Q}_i(q)}{t^{PS}_{cl(i)}} \right\rceil - 1\right) t^{PS}_{cl(i)} + \phi^{PS}_{cl(i)}}_{\text{(a)}}\right) + I^{HPB}_i(\tilde{Q}_i(q)) \tag{4.44}$$

Proof. Follows the proofs of Theorems 9 and Theorem 12 with Lemmata 16, 17, and 18. The construct of splitting the computation of the queueing delay in Eq. (4.43) has been addressed in the proof of Theorem 30. To compute an upper bound on the multiple-event queueing delay for the q-th activation of PS traffic stream i, we follow the argumentation of Theorem 12 and conservatively assume that the q-th frame arrives at the latest point in time at which it will be part of the current queueing delay. This is modeled by term (a). In IEEE 802.1Qch, this point is determined by the PS interval pattern of PS class $cl(i)$, which are aligned to the end of class $cl(i)$'s PS offset (cf. Definition 30). Particularly, it depends on the number of PS intervals of PS class $cl(i)$ which end during $\tilde{Q}_i(q)$. For any time interval $\tilde{Q}_i(q)$, the maximum number of PS intervals of PS class $cl(i)$, which end in this $\tilde{Q}_i(q)$, can be computed by $\lceil \tilde{Q}_i(q)/t^{PS}_{cl(i)} \rceil$. Multiplying this number by the PS interval length of PS class $cl(j)$ yields the time interval during which the interference from same-priority PS streams can be accumulated. However, since the first interval (only) contributes with time $\phi^{PS}_{cl(i)}$ (cf. Definition 30), we need to subtract one from the number of intervals, which end in $\tilde{Q}_i(q)$, and add $\phi^{PS}_{cl(i)}$ instead in order to collect the overall interference from interfering same priority PS traffic streams. □

With the help of the lower-, same-, and higher priority blocking terms, also, the multiple-event scheduling horizon for PS streams can be derived.

Theorem 32. *The multiple-event scheduling horizon for the q-th frame arrival of a PS traffic stream i in IEEE 802.1Qch is upper bounded by*

$$S_i(q) = \tilde{S}_i(q) + \phi^{PS}_{cl(i)} \tag{4.45}$$

where

$$\tilde{S}_i(q) = I^{LPB}_i + \tilde{I}^{SPB}_i(q, \tilde{S}_i(q)) + \tilde{I}^{HPB}_i(\tilde{S}_i(q)) \tag{4.46}$$

with

$$\tilde{I}^{SPB}_i(q, \Delta t) = qC^+_i + \sum_{j \in fifo(i)} \eta^+_j \Bigg(\underbrace{\left(\left\lceil \frac{\Delta t}{t^{PS}_{cl(j)}} \right\rceil - 1 \right) t^{PS}_{cl(j)} + \phi^{PS}_{cl(i)}}_{\text{(a)}} \Bigg) C^+_j \tag{4.47}$$

and

$$\tilde{I}^{HPB}_i(\Delta t) = \sum_{j \in \{k | k \in hp(i) \wedge cl(k) \notin \mathcal{C}\}} \eta^+_j(\Delta t)\, C^+_j + \sum_{j \in \{k | k \in hp(i) \wedge cl(k) \in \mathcal{C}\}} \eta^+_j \left(\left\lceil \frac{\Delta t}{t^{PS}_{cl(j)}} \right\rceil t^{PS}_{cl(j)} \right) C^+_j \tag{4.48}$$

Proof. The scheduling horizon of the q-th frame arrival of PS traffic stream i ends, when all q frames of stream i have been transmitted and all frames of interfering traffic streams that arrived within the scheduling horizon of the q-th frame arrival of PS traffic stream i have been transmitted as well. Only then an additional hypothetical activation of infinitesimal small execution time could be processed. The amount of frames from interfering traffic streams can be bounded by Lemmata 16, 17, and 18. However, similar to Theorem 20, we have to adjust the blocking terms from Eqs. (4.4) and (4.33) with Eqs. (4.47) and (4.48) to address the fact that the scheduling horizon is defined over a right half-open interval, i.e. we use $\eta^+_j(\Delta t)$ instead of $\eta^{+]}_j(\Delta t)$. Additionally, we need to consider all q frame arrivals when computing the multiple-event scheduling horizon for the q-th frame arrival of traffic stream i. This is also addressed by Eq. (4.47). The argumentation for the time interval represented by term (a) in Eq. (4.47) follows the proof of Theorem 31. □

The worst-case frame forwarding times and maximum buffer size requirements for PS traffic streams in IEEE 802.1Qch can be computed as described in Section 4.3.2 and Section 4.3.4.

As discussed, in IEEE 802.1Qch, frames of traffic streams of a PS class I that arrive during a given PS interval of PS class I are not released before the end of this PS interval. Consequently, frames of a PS traffic stream $i \in I$ under analysis that arrive after the last full PS interval of stream i's scheduling horizon do not contribute to this scheduling horizon anymore, as the corresponding output port is idle before these frames are considered by its transmission selection mechanism. Instead, these frames start a new scheduling horizon. We can exploit is to derive a tighter bound on the stopping condition.

Theorem 33. *The maximum number $\hat{q}_i$ of frame arrivals in the scheduling horizon of a PS traffic stream i equals the maximum number of frames that arrive during the scheduling horizon of their respective predecessors:*

$$\hat{q}_i = \max_{q \geq 1} \left\{ q \;\middle|\; \delta_i^-(q) \leq \left(\left\lceil \frac{S_i(q-1) - \phi^{PS}_{cl(i)}}{t^{PS}_{cl(i)}} \right\rceil - 1 \right) t^{PS}_{cl(i)} + \phi^{PS}_{cl(i)} \right\} \tag{4.49}$$

Proof. Frame arrivals after the last full PS interval cannot contribute to the scheduling horizon. The argumentation follows the proof of Theorem 30 (term (a) in Eq. (4.44) in particular) with $S_i(q-1) - \phi^{PS}_{cl(i)} = \tilde{S}_i(q-1)$ (cf. Eq. (4.45)). □

In Figure 4.5, for example, frame $I.4$, even though it arrives within the multiple-event scheduling horizon started by the arrival of frame $I.1$, is not part of this scheduling horizon, as the output port is idle before $I.4$ can be scheduled (i.e. before t_2).

Theorem 34. *The bound on $\hat{q}_i$ in Eq. (4.49) is tighter than or equal to the bound in Eq. (3.11).*

Proof. We show this by contradiction. Assume that Eq. (4.49) is not tighter than or equal to the bound in Eq. (3.11), i.e.

$$\left(\left\lceil \frac{S_i(q-1) - \phi^{PS}_{cl(i)}}{t^{PS}_{cl(i)}} \right\rceil - 1 \right) t^{PS}_{cl(i)} + \phi^{PS}_{cl(i)} > S_i(q-1)$$

Subtracting $\phi^{PS}_{cl(i)}$ followed by division of $t^{PS}_{cl(i)} > 0$ (cf. Definition 29) yields

$$\left\lceil \frac{S_i(q-1) - \phi^{PS}_{cl(i)}}{t^{PS}_{cl(i)}} \right\rceil - 1 > \frac{S_i(q-1) - \phi^{PS}_{cl(i)}}{t^{PS}_{cl(i)}}$$

which is a contradiction. Hence, the theorem follows. □

Note that the upper bound on the number of frame arrivals presented in Definition 15 can also be used, since it is a conservative bound. It might, however, consider too many frame arrivals, so that $\hat{q}_i$ might be unnecessarily large or does not exist, especially in high-load scenarios.

So far, the presented analysis for PS traffic streams in IEEE 802.1Qch depends on the PS offset. The number of PS offsets to investigate to find the longest multiple-event queueing delay, the multiple-event processing time, and the multiple-event scheduling horizon can be reduced to one.

Theorem 35. *The multiple-event queueing delay, the multiple-event processing time, and the multiple-event scheduling horizon are maximized if $\phi^{PS}_{cl(i)} = t^{PS}_{cl(i)}$.*

Proof. Follows directly from Eqs. (4.40), (4.42), and (4.45) and the fact that, the PS interval pattern repeats periodically. □

A formal comparison of the worst-case guarantees of IEEE 802.1Qch and IEEE 802.1Q shows no benefit of IEEE 802.1Qch over IEEE 802.1Q.[10]

Corollary 2. *IEEE 802.1Qch becomes IEEE 802.1Q if all t_I^{PS} approach 0.*

Proof. Follows from Lemmata 12, 16, 13, 13, and 18, as well as the comparison of Eqs. (4.31) and (4.32) with Eq. (4.5). □

Theorem 36. *The worst-case bounds (of the multiple-event queueing delay, multiple-event processing time, multiple-event scheduling horizon, and frame forwarding time) for IEEE 802.1Qch are always worse or equal to the bounds of IEEE 802.1Q.*

Proof. From Corollary 2 we know that IEEE 802.1Qch becomes IEEE 802.1Q if all t_I^{PS} approach 0. We now show that the worst-case bounds approach the bounds of IEEE 802.1Q from Section 4.3.

For non-PS traffic streams, the higher-priority interference from PS traffic streams according to Corollary 1 (with Eq. (4.32) in particular) is always greater or equal than the interference from IEEE 802.1Q, as $\forall t_{cl(i)}^{PS} > 0 : \lceil \Delta t / t_{cl(i)}^{PS} \rceil t_{cl(i)}^{PS} \geq \Delta t$.

For PS traffic streams, this also holds for the higher-priority interference from PS traffic streams and also applies to the same-priority interference of the multiple-event processing time in Eq. (4.40) and the multiple-event scheduling horizon in Eq. (4.47). Additionally, the multiple-event queueing delay, multiple-event processing time, and multiple-event scheduling horizon of PS traffic streams are delayed by the length of t_I^{PS} (cf. Eqs. (4.40), (4.43), and (4.45) with Theorem 35).

As the worst-case frame forwarding time is derived from the multiple-event queueing delay, the theorem applies to the worst-case frame forwarding time as well. □

4.5.2 Design Considerations

In order for IEEE 802.1Qch to perform as intended, i.e. to bound the residence time of PS frames in switches via cyclic frame forwarding, the PS interval length of each PS class must be long enough so that all PS frames that arrive during one PS interval can be transferred within the next one without any frame transmission exceeding the intervals. A conservative check is to determine if, for all PS classes $I \in \mathcal{C}$, the maximum multiple-event processing time (measured from the start of the corresponding PS interval) for each PS traffic stream stream $i \in I$ is smaller than I's PS interval length.

$$\forall i \in \bigcup_{I \in \mathcal{C}} I : B_i^+(\hat{q}_i) - t_{cl(i)}^{PS} \leq t_{cl(i)}^{PS} \tag{4.50}$$

If Eq. (4.50) is not met, the key goal of IEEE 802.1Qch to compute path latency solely as a function of the number of hops would be missed.

[10] Please note that this is a rather academic comparison. IEEE 802.1Q and IEEE 802.1Qch have very different design goals and IEEE 802.1Qch, if configured correctly, has a right to exist for the reasons outlined in Section 2.2.3.3.

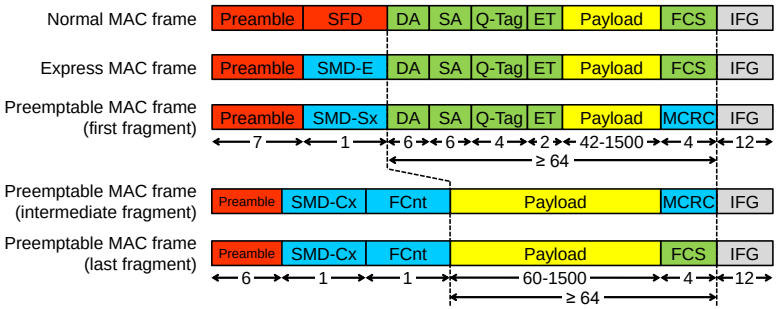


Figure 4.6: MAC frame formats: original IEEE 802.3 MAC frame format on the top followed by four IEEE 802.3br frame formats (all sizes in bytes).

4.6 Analysis of Frame Preemption (IEEE 802.3br)

As discussed in Section 2.2.3.4, IEEE 802.3br adds frame preemption to Ethernet. It defines two MAC interfaces, one (non-preemptable) express MAC and one preemptable MAC and each Ethernet traffic class is mapped to either of these interfaces. IEEE 802.3br only allows one level of preemption, i.e. only frames of express traffic classes can preempt frames of preemptable traffic classes. Neither can frames of express traffic classes preempt frames of other express traffic classes nor can frames of preemptable traffic classes preempt frames of other preemptable traffic classes, regardless of their priority.

As a result of preemption, a frame is split into two or more fragments. One design goal of IEEE 802.3br was to preserve the basic structure of IEEE 802.3's MAC frame format in order to make frame preemption transparent to Ethernet's physical layer (PHY). Hence, each fragment must appear to the PHY as a valid Ethernet frame. To this end, IEEE 802.3br defines different MAC frame formats (see Figure 4.6).

The MAC frames of IEEE 802.3br's dual MAC interface are also called MFrames (MAC merge frames). The express frame format is very similar to the IEEE 802.3 MAC frame format. The only difference is that the SFD is replaced by an SMD-E (start MFrame delimiter - express) field to signal the express frame. The first fragment of a preemptable frame is also very similar to the express frame. Its start, however, is signaled by the SMD-Sx (SMD - start fragment) delimiter. If this frame is not preempted, it is terminated by an FCS followed by an IFG (not shown in Figure 4.6). A preemptable frame may be split into fragments. With the exception of the last one, each fragment is terminated by an MCRC (instead of an FCS) followed by an IFG. This MCRC protects the fragment's data and is necessary to let the fragment appear as a valid IEEE 802.3 MAC frame to an Ethernet PHY. In the last fragment, this MCRC is replaced by the FCS of the entire original frame, i.e. if it had been transmitted without preemption. All fragments following the first one are signaled by SMD-Cx (SMD - continuation fragment) and a fragment counter (FCnt). As IEEE 802.3br only allows one level of preemption, DA, SA, Q-Tag,

and ET only have to be transmitted once. Thus, all fragments after the first one can accommodate a slightly larger payload. SMD-Sx, SMD-Cx, and FCnt are implemented as modulo 4 counters with a hamming distance of four to protect them against bit errors. Hence, apart from signaling frame and fragment starts, they are used to detect lost fragments.

All MAC frames (including fragments) must meet the minimum Ethernet frame size requirement, which is 84 bytes (including IFG, Figure 4.6). If the actual payload is too small to meet this requirement, the payload field is padded accordingly. However, preemption is not allowed to introduce any additional padding to the resulting fragments. This imposes constraints on the preemption instant and the fragmentation granularity: (1) A preemptable frame cannot be preempted until the fragment (of it) that is currently being transmitted fulfills the minimum frame size requirement. (2) A preemptable frame or its remaining data can only be fragmented further if the resulting fragments meet the minimum frame size requirement.

4.6.1 Illustration of Frame Preemption

Figure 4.7 illustrates frame preemption in IEEE 802.1Q inside a switch port. There are two frames: *HP* and *LP*, where *HP* has a higher priority than *LP*. Frame *HP* arrives while *LP* is in transmission. In the non-preemptive scenario, *HP* has to wait for *LP* to finish its transmission before it can be sent. This results in a long transmission latency R^{+}_{HP} of *HP*.

Under frame preemption, the transmission of frame *LP* can be preempted to allow the higher-priority frame *HP* to be sent earlier. As preemption entails a certain overhead, i.e. the first fragment of frame *LP* must be terminated with an MCRC followed by an inter frame gap, *HP* cannot be sent immediately. After *HP* has been transmitted, *LP* is resumed. Again, this introduces a certain overhead by prepending the new fragment of *LP* with a preamble and other information, which is required to reassemble the preempted frame at the receiving end. As can be seen, preemption decreases the delay that *HP* experiences. Observe, however, that the preemption overhead increases the time during which the switch port is busy transmitting data.

4.6.2 Resource-Level Analysis of IEEE 802.3br

In the following two sections, we extend the resource-level analyses for IEEE 802.1Q and IEEE 802.1Qbv to support frame preemption according to IEEE 802.3br. This entails a redefinition of IEEE 802.1Q's and IEEE 802.1Qbv's blocking terms as well as their multiple-event scheduling horizons and multiple-event processing times. First, however, we introduce a set of common definitions and lemmata.

We distinguish two sets of traffic classes.

Definition 31 (Express Traffic Classes, Express Traffic Streams, Express Frames)**.** *Let $\mathcal{E}$ be the set of non-preemptable express traffic classes. The traffic streams that belong to an express traffic class are called express traffic streams and the frames of express traffic streams are called express frames.*

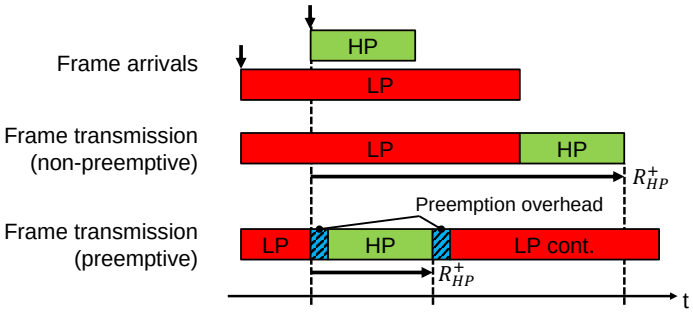


Figure 4.7: Example of non-preemptive and preemptive frame transmission according to IEEE 802.3br for IEEE 802.1Q

Definition 32 (Preemptable Traffic Classes, Preemptable Traffic Streams, Preemptable Frames)**.** *Let $\mathcal{P}$ be the set of preemptable traffic classes. The traffic streams that belong to a preemptable traffic class are called preemptable traffic streams and the frames of preemptable traffic streams are called preemptable frames.*

As discussed, there is only one level of preemption. Express frames cannot preempt each other regardless of their priority and also preemptable frames cannot preempt each other regardless of their priority. Only preemptable frames can be preempted by express frames. As express traffic is considered to be more critical than preemptable traffic, we assume that all express traffic streams have a higher Ethernet priority than preemptable streams.

In order to derive upper bounds on the timing behavior, it is essential to determine the longest frame or fragment, which can block an express frame, and the preemption overhead per fragment.

Lemma 19. *The longest lower-priority frame or fragment that can block an express frame is 143 bytes long.*

Proof. Two cases must be considered: (1) a lower-priority frame that cannot be preempted and (2) a frame's last fragment that cannot be preempted further.

For both cases, we first determine the smallest frame or fragment that can still be preempted. (1) As we are interested in the smallest frame, only one preemption is possible. If we split a frame, both resulting fragments must fulfill the minimum Ethernet frame size requirement of 84 bytes (preemption is not allowed to introduce additional padding). Hence, our goal is to minimize the combined payload $p = p_1 + p_2$ of both fragments, s.t. Preamble + SMD-Sx + DA + SA + Q-Tag + ET + p_1 + MCRC + IFG ≥ 84 bytes (first fragment) and Preamble + SMD-Cx + FCnt + p_2 + FCS + IFG ≥ 84 bytes (second fragment) (see Figure 4.6). This yields $p_1 = 42$ and $p_2 = 60$ bytes. Hence, the smallest frame, which can still be preempted, must have a payload of $p = 102$ bytes, which results in an overall frame size of 144 bytes, if Preamble, SMD-Sx, DA, SA, Q-Tag, ET, FCS/MCRC, and IFG are added.

(2) Any remaining data can only be split further, if the resulting fragments fulfill the minimum Ethernet frame size requirement of 84 bytes. Here we minimize $p = p_1 + p_2$ s.t. Preamble + SMD-Cx + FCnt + p_1 + MCRC + IFG $\geq$ 84 bytes and Preamble + SMD-Cx + FCnt + p_2 + FCS + IFG ≥ 84 bytes, which yields $p_1 = p_2 = 60$ bytes. Hence, the smallest amount of remaining data p that can still be preempted is 120 bytes and results in an overall fragment size of 144 bytes, if Preamble, SMD-Cx, FCnt, FCS, and IFG are added.

In both cases, we get the largest frame/fragment that cannot be preempted further by subtracting 1 byte from the payload p, yielding frames/fragments of 143 bytes. □

Lemma 20. *The overhead per preemption is 24 bytes.*

Proof. As each additional fragment requires an additional Preamble, SMD-Cx, FCnt, MCRC, and IFG, the preemption overhead of a single preemption is 24 bytes (see Figure 4.6). □

In order to support IEEE 802.3br frame preemption, we have to adjust the formal definition of the multiple-event queueing delay from Definition 20.

Definition 33 (Multiple-Event Queueing Delay under Frame Preemption)**.** *The multiple-event queueing delay $Q_{i,a}(q)$ of the q-th frame of a traffic stream i, which arrived at time a relative to the beginning of the multiple-event queueing delay, is the longest time interval from the arrival of the first frame from traffic stream i until the last non-preemptable part of the q-th frame of traffic stream i is considered by the transmission selection mechanism.*

For express traffic this *part* is the frame itself. For preemptable traffic it is a minimum-sized (non-preemptable) fragment. This is because a preemptable frame can still be preempted until the transmission of its last non-preemptable fragment has started. If this last fragment is minimum-sized, the time for interference is maximized.

4.6.2.1 Resource-Level Analysis of IEEE 802.1Q with IEEE 802.3br

The analysis closely follows the analysis of IEEE 802.1Q, but extends its blocking terms to include the timing effects of frame preemption. We present one resource-level analysis for express traffic in IEEE 802.1Q and one for preemptable traffic in IEEE 802.1Q.

4.6.2.1.1 Resource-Level Analysis of Express Traffic

The lower-priority blocking for express traffic streams can potentially be reduced by taking frame preemption into account.

Lemma 21 (Lower-Priority Blocking of Express Traffic in IEEE 802.3br for IEEE 802.1Q)**.** *The lower-priority blocking an express traffic stream i can experience in IEEE 802.3br for IEEE 802.1Q is bounded by*

$$I_i^{LPB} = \max\left\{ \underbrace{\max_{j \in lp^E(i)} \{C_j^+\}}_{\text{(a)}}, \min\left\{ \underbrace{\max_{j \in lp^P(i)} \{C_j^+\}}_{\text{(b)}}, \underbrace{\frac{143\,\text{bytes}}{r_{TX}}}_{\text{(c)}} \right\} \right\} \tag{4.51}$$

where $lp^E(i)$ is the set of express traffic streams of lower priority than stream i and $lp^P(i)$ is the set of preemptable traffic streams of lower priority than stream i.

Proof. As express frames cannot preempt each other, the longest blocking time from lower-priority express frames is given by term (a). The longest blocking time from lower-priority preemptable frames can be derived from the longest non-preemptable fragment. This fragment has a size of 143 bytes (Lemma 19), which can be translated to a blocking time by dividing by the link speed r_{TX} in term (c). If all lower-priority preemptable frames are shorter than 143 bytes, this can be exploited by taking the minimum of the largest lower-priority preemptable frame (term (b)) and term (c). We do not know which term (term (a) or the minimum over terms (b) and (c)) is larger. Thus, we take the maximum. □

The same-priority blocking for express traffic streams can be bounded as in IEEE 802.1Q.

Lemma 22 (Same-Priority Blocking of Express Traffic Streams in IEEE 802.3br for IEEE 802.1Q)**.** *The same-priority blocking the q-th frame of an express traffic stream i that arrived at time a can experience can be bounded as for IEEE 802.1Q in Eq. (4.4).*

Proof. As frames of express traffic streams cannot preempt themselves, the proof follows the proof of Lemma 2. □

The higher-priority blocking for express traffic streams can be bounded as in IEEE 802.1Q.

Lemma 23 (Higher-Priority Blocking of Express Traffic Streams in IEEE 802.3br for IEEE 802.1Q)**.** *The higher-priority blocking an express traffic stream i can experience in any time interval Δt in IEEE 802.3br for IEEE 802.1Q is bounded as for IEEE 802.1Q in Eq. (4.5).*

Proof. We assumed that all express traffic classes have a higher Ethernet priority than preemptable traffic classes. Consequently, the higher-priority blockers of an express traffic streams are also express traffic streams. As an express traffic stream can be blocked by all higher-priority traffic streams, the proof follows the proof of Lemma 3. □

With the help of these blocking terms the multiple-event queueing delay can be bounded.

Theorem 37. *The multiple-event queueing delay in IEEE 802.3br for IEEE 802.1Q for the q-th frame arrival of express traffic stream i that arrived at time a is upper bounded by*

$$Q_{i,a}(q) = I_i^{LPB} + I_i^{SPB}(q, a) + I_i^{HPB}\left(Q_{i,a}(q)\right) \tag{4.52}$$

Proof. The multiple-event queueing delay must take all blocking effects into account. From Lemmata 21, 22, and 23 the theorem follows (cf. proof of Theorem 18). □

With the multiple-event queueing delay, the multiple-event processing time can be bounded.

Theorem 38. *The multiple-event processing time for the q-th frame arrival of express traffic stream i in IEEE 802.3br for IEEE 802.1Q is upper bounded by*

$$B_i^+(q) = Q_i(q) + C_i^+ \tag{4.53}$$

where $Q_i(q)$ is an upper bound on the multiple-event queueing delay for the q-th frame arrival of express traffic stream i:

$$Q_i(q) = I_i^{LPB} + I_i^{SPB}\left(q, Q_i(q)\right) + I_i^{HPB}\left(Q_i(q)\right) \tag{4.54}$$

Proof. Follows the proofs of Theorems 9 and 12 (cf. proof of Theorem 19). □

With the help of the blocking terms in Lemmata 21, 22, and 23, also, the multiple-event scheduling horizon can be derived.

Theorem 39. *The multiple-event scheduling horizon for the q-th frame arrival of express traffic stream i in IEEE 802.3br for IEEE 802.1Q is upper bounded by*

$$S_i(q) = I_i^{LPB} + \tilde{I}_i^{SPB}(q, S_i(q)) + \tilde{I}_i^{HPB}\left(S_i(q)\right) \tag{4.55}$$

with

$$\tilde{I}_i^{SPB}(q, \Delta t) = qC_i^+ + \sum_{j \in fifo(i)} \eta_j^+(\Delta t) C_j^+ \tag{4.56}$$

and

$$\tilde{I}_i^{HPB}(\Delta t) = \sum_{j \in hp(i)} \eta_j^+(\Delta t) C_j^+ \tag{4.57}$$

Proof. Follows the proof of Theorem 20. □

The worst-case frame forwarding times and maximum buffer size requirements for express traffic streams in IEEE 802.3br for IEEE 802.1Q can be computed as described in Section 4.3.2 and Section 4.3.4.

4.6.2.1.2 Resource-Level Analysis of Preemptable Traffic

The lower-priority blocking for preemptable traffic streams can be bounded as in IEEE 802.1Q.

Lemma 24 (Lower-Priority Blocking of Peemptable Traffic in IEEE 802.3br for IEEE 802.1Q)**.** *The lower-priority blocking a preemptable traffic stream i can experience in IEEE 802.3br for IEEE 802.1Q is bounded as for IEEE 802.1Q in Eq. (4.3).*

Proof. The worst-case lower-priority blocking for a preemptable frame of traffic stream i occurs when the largest frame of a traffic stream with lower-priority than stream i starts transmitting right before the first frame of stream i starts transmitting (cf. Lemma 1). □

In order to bound the same-priority blocking for preemptable traffic streams, we have to take into account that preemptable frames can be preempted and hence potentially experience more blocking (i.e. not only until the last frame starts transmission, but until the last non-preemptable fragment starts transmission (cf. Definition 33)).

Lemma 25 (Same-Priority Blocking of Preemptable Traffic Streams in IEEE 802.3br for IEEE 802.1Q)**.** *The same-priority blocking the q-th frame of a preemptable traffic stream i that arrived at time a can experience in IEEE 802.3br for IEEE 802.1Q is bounded by*

$$I_i^{SPB}(q,a) = \begin{cases} (q-1)C_i^+ + C_i^+ - \frac{84\,\text{bytes}}{r_{TX}} + \sum\limits_{j \in fifo(i)} \eta_j^{+]}(a) C_j^+ & \text{if } C_i^+ > \frac{143\,\text{bytes}}{r_{TX}} \\ (q-1)C_i^+ + \sum\limits_{j \in fifo(i)} \eta_j^{+]}(a) C_j^+ & \text{if } C_i^+ \leq \frac{143\,\text{bytes}}{r_{TX}} \end{cases} \tag{4.58}$$

Proof. We have to distinguish two cases: (1) the frames of preemptable traffic stream i are too small to be preempted, i.e. $C_i^+ \leq \frac{143\,\text{bytes}}{r_{TX}}$ (cf. Lemma 19) and (2) the frames of preemptable traffic stream i can be preempted , i.e. $C_i^+ > \frac{143\,\text{bytes}}{r_{TX}}$.

Case (1) (bottom alternative of Eq. (4.58)): The q-th arrival of a frame of preemptable traffic stream i, which arrived at time a, has to wait for its own $q-1$ (in this case non-preemptable) predecessors to finish and for all frames of other same-priority preemptable traffic streams, which have arrived previous to its arrival. Hence, the proof follows from proof of Lemma 2.

Case (2) (top alternative of Eq. (4.58)): In this case the frames of preemptable traffic stream i are preemptable. So, additionally, the q-th frame of preemptable traffic stream i may be preempted by express traffic (of higher priority) until the transmission of its last non-preemptable fragment has started. To maximize the potential number of preemptions, this last fragment is, in the worst-case, of minimum size (i.e. 84 bytes), leaving the rest of the size of the q-th frame (i.e. $C_i^+ - \frac{84\,\text{bytes}}{r_{TX}}$) for preemption.

The amount of same-priority blocking is maximized, if the critical instant scenario is chosen such that frames of same-priority traffic streams start

arriving as fast as possible together with the first frame of traffic stream i. In the worst-case, frames of same-priority preemptable traffic streams arrive just at the end of a. We include the interference from these frames by using the closed-interval upper arrival function, i.e. assuming that these frame arrive just before the q-th frame of stream i at a. □

The higher-priority blocking for preemptable traffic streams can be bounded as in IEEE 802.1Q.

Lemma 26 (Higher-Priority Blocking of Preemptable Traffic Streams in IEEE 802.3br for IEEE 802.1Q)**.** *The higher-priority blocking a preemptable traffic stream i can experience in any time interval Δt in IEEE 802.3br for IEEE 802.1Q is bounded as for IEEE 802.1Q in Eq. (4.5).*

Proof. As a preemptable traffic stream can be blocked by all higher-priority traffic streams, the proofs follows the proof of Lemma 3. □

When this higher-priority blocking is due to preemption, there is a certain preemption overhead. If the number of preemptions within a frame's queueing delay is known, their overhead can be modeled as an additional blocking term, extending the queueing delay. We start by deriving an upper bound on the maximum number of preemptions per frame.

Lemma 27. *The maximum number of preemptions of a single frame with payload p_i^+ of a preemptable traffic stream i is given by*

$$F_i^+ = \left\lfloor \frac{p_i^+ - 42\,\text{bytes}}{60\,\text{bytes}} \right\rfloor \tag{4.59}$$

Proof. The number of preemptions of a frame of preemptable stream i is maximized, if we distribute the frame's maximum payload p_i^+ among as many fragments as possible. In IEEE 802.3br, the minimum payload of the first fragment of a preempted frame is 42 bytes (see Figure 4.6). All following (non-initial) fragments must carry a minimum payload of 60 bytes (see Figure 4.6). Thus, we can compute the maximum number of preemptions of a single frame by first subtracting 42 bytes from its maximum payload p_i^+ and then dividing the remaining payload by 60 bytes. If $p_i^+ - 42$ is not divisible by 60 without remainder, one of the fragments must be larger and also carry the otherwise remaining $x < 60$ bytes, as IEEE 802.3br does not allow fragments smaller than the minimum Ethernet frame size requirement. Hence, we round down. □

The maximum number of frame preemptions in the entire queueing delay can be computed by multiplying the number of preemptable frames in the queueing delay by their respective F_i^+. The following lemmata derive upper bounds on the number of frame preemptions within the queueing delay for the respective blocking contexts.

Lemma 28. *The number of frame preemptions from lower-priority preemptable traffic streams that can block frames of a preemptable traffic stream i within its multiple-event queueing delay is bounded by*

$$N_i^{LP} = \max_{j \in lp^P(i)} \left\{ F_j^+ \right\} \tag{4.60}$$

Proof. Follows the proof from Lemma 24, i.e. a preemptable frame of traffic stream i can be blocked by at most one lower-priority preemptable frame. □

As preemptable frames cannot preempt each other, this lower-priority frame will be transmitted entirely, but may be fragmented due to preemptions from express traffic.

Lemma 29. *The number of frame preemptions from same-priority preemptable traffic streams that can block the transmission of q frames of a preemptable traffic stream i within its multiple-event queueing delay is bounded by*

$$N_i^{SP}(q,a) = \begin{cases} \underbrace{qF_i^+ - 1}_{\text{(a)}} + \sum\limits_{j \in fifo(i)} \eta_j^{+]}(a) F_j^+ & \text{if } C_i^+ > \frac{143\,\text{bytes}}{r_{TX}} \\ \sum\limits_{j \in fifo(i)} \eta_j^{+]}(a) F_j^+ & \text{if } C_i^+ \leq \frac{143\,\text{bytes}}{r_{TX}} \end{cases} \tag{4.61}$$

where a is the arrival time of the q-th frame of preemptable traffic stream i.

Proof. Follows the from Lemma 25, including the differentiation of two cases. Similar to case (2) of Eq. (4.58), we do not consider the preemption overhead of the last fragment (term (a)).[11] □

Lemma 30. *The number of frame preemptions from higher-priority preemptable traffic streams that can block the transmission of q frames of a preemptable traffic stream i in any time interval Δt is bounded by*

$$N_i^{HP}(\Delta t) = \sum_{j \in hp^P(i)} \eta_j^{+]}(\Delta t) F_j^+ \tag{4.62}$$

where $hp^P(i)$ is the set of preemptable traffic streams with a higher priority than preemptable traffic stream i.

Proof. Follows the from Lemma 26, i.e. within any time interval of length Δt, the number of preemptable frames of higher priority than stream i can be computed by summing up the higher-priority frame arrivals over Δt. □

[11] This fragment (and its preemption overhead) will be considered later when we derive the worst-case frame forwarding time of the q-th frame of preemptable traffic stream i.

Eqs. (4.60), (4.61), and (4.62) bound the maximum number of frame preemptions a frame of a preemptable traffic stream i can experience during its queueing delay by assuming that all frames are split into their maximum number of fragments. However, the number of frame preemptions can also be bounded by the actual (worst-case) preemption pattern from higher-priority express traffic. The following lemma computes a bound on the preemption overhead considering both bounds.

Lemma 31 (Preemption Overhead in IEEE 802.3br for IEEE 802.1Q)**.** *The preemption overhead in a time interval of length Δt containing q frames of the traffic stream under analysis i, out of which the q-th one arrived at time a relative to the beginning of Δt, is upper bounded by:*

$$I_i^{PO}(\Delta t, q, a) = \frac{24\,\text{bytes}}{r_{TX}} \min \left\{ \sum_{j \in hp^E(i)} \eta_j^{+]}(\Delta t),\ N_i^{LP} + N_i^{SP}(q,a) + N_i^{HP}(\Delta t) \right\} \quad (4.63)$$

where $hp^E(i)$ is the set of express traffic streams of higher priority than preemptable stream i.

Proof. From Lemma 20 we know that the preemption overhead is 24 bytes. Its duration can be derived by dividing it by the link speed r_{TX}. Given Δt, q, and a, the maximum number of frame preemptions can be computed by summing up Eqs. (4.60), (4.61), and (4.62) (second term in minimum). In the worst-case each higher-priority express frame causes a preemption. Hence, there can be no more frame preemptions than there are higher-priority express frames within Δt (first term in minimum). So, we take the minimum of both terms. □

With the help of the lower-, same-, and higher-priority bocking terms as well as the preemption overhead the multiple-event queueing delay can be bounded.

Theorem 40. *The multiple-event queueing delay in IEEE 802.3br for IEEE 802.1Q for the q-th frame arrival of preemptable traffic stream i that arrived at time a is upper bounded by*

$$Q_{i,a}(q) = I_i^{LPB} + I_i^{SPB}(q,a) + I_i^{HPB}(Q_{i,a}(q)) + I_i^{PO}(Q_{i,a}(q), q, a) \quad (4.64)$$

Proof. The multiple-event queueing delay must take all blocking effects and the preemption overhead into account. From Lemmata 24, 25, 26, and 31 the theorem follows (cf. proof of Theorem 18). □

With the multiple-event queueing delay, the multiple-event processing time can be bounded.

Theorem 41. *The multiple-event processing time for the q-th frame arrival of preemptable traffic stream i in IEEE 802.3br for IEEE 802.1Q is upper bounded by*

$$B_i^+(q) = Q_i(q) + C_i^+ \tag{4.65}$$

where $Q_i(q)$ is an upper bound on the multiple-event queueing delay for the q-th frame arrival of preemptable traffic stream i:

$$Q_i(q) = I_i^{LPB} + I_i^{SPB}\left(q, Q_i(q)\right) + I_i^{HPB}\left(Q_i(q)\right) + I_i^{PO}\left(Q_i(q), q, Q_i(q)\right) \tag{4.66}$$

Proof. Follows the proofs of Theorems 9 and 12 (cf. proof of Theorem 19) with Theorem 40. □

With the help of the blocking terms in Lemmata 24, 25, and 26 as well as the preemption overhead from Lemma 31, also, the multiple-event scheduling horizon can be derived.

Theorem 42. *The multiple-event scheduling horizon for the q-th frame arrival of express traffic stream i in IEEE 802.3br for IEEE 802.1Q is upper bounded by*

$$S_i(q) = I_i^{LPB} + \tilde{I}_i^{SPB}(q, S_i(q)) + \tilde{I}_i^{HPB}\left(S_i(q)\right) + \tilde{I}_i^{PO}\left(q, S_i(q)\right) \tag{4.67}$$

with

$$\tilde{I}_i^{SPB}(q, \Delta t) = qC_i^+ + \sum_{j \in fifo(i)} \eta_j^+(\Delta t) C_j^+ \tag{4.68}$$

and

$$\tilde{I}_i^{HPB}(\Delta t) = \sum_{j \in hp(i)} \eta_j^+(\Delta t) C_j^+ \tag{4.69}$$

and

$$\tilde{I}_i^{PO}(q, \Delta t) = \frac{24\,\text{bytes}}{r_{TX}} \min\left\{ \sum_{j \in hp^E(i)} \eta_j^+(\Delta t),\ N_i^{LP} + \tilde{N}_i^{SP}(q, \Delta t) + \tilde{N}_i^{HP}(\Delta t) \right\} \tag{4.70}$$

where

$$\tilde{N}_i^{SP}(q, \Delta t) = qF_i^+ + \sum_{j \in fifo(i)} \eta_j^+(\Delta t) F_j^+ \tag{4.71}$$

and

$$\tilde{N}_i^{HP}(\Delta t) = \sum_{j \in hp^P(i)} \eta_j^+(\Delta t) F_j^+ \tag{4.72}$$

Proof. Follows the proof of Theorem 20, but we have to consider the additional preemption overhead with Eq. (4.70). Similar to the same- and higher-priority blocking in Eqs. (4.68) and (4.69), we have to consider all q frames when computing the multiple-event scheduling horizon and also use the half-open versions of the event arrival functions (cf. Theorem 20). Note that, as all timing effects of frame preemption are captured by Eq. (4.70), we do not need to distinguish different cases as we did for the same-priority blocking when we derived bounds on the queueing delay. This is because, for the scheduling horizon, only the amount of released workload (frames) is important, but not in which order these frames have been transmitted or when (and if) they have been preempted (cf. [100]). Also note that in contrast to Eq. (4.61), we do not need to distinguish between preemptable and non-preemptable frames, as for non-preemptable frames the term qF_i^+ evaluates to 0. □

The maximum buffer size requirements for preemptable traffic streams in IEEE 802.3br for IEEE 802.1Q can be computed as described in Section 4.3.4. To compute the worst-case frame forwarding times, however, we have to take fragmentation into account.

Theorem 43. *Under IEEE 802.3br for IEEE 802.1Q, the worst-case response time of the q-th frame of a preemptable traffic stream i that arrived at time a is upper bounded by*

$$R_i(q) = \begin{cases} \max\limits_{a \in A_{q,i}} \left\{ Q_{i,a}(q) + \frac{84\,\text{bytes}}{r_{TX}} - a \right\} & \text{if } C_i^+ > \frac{143\,\text{bytes}}{r_{TX}} \\ \max\limits_{a \in A_{q,i}} \left\{ Q_{i,a}(q) + C_i^+ - a \right\} & \text{if } C_i^+ \leq \frac{143\,\text{bytes}}{r_{TX}} \end{cases} \tag{4.73}$$

Proof. Follows the proof of Theorem 15, i.e. from the queueing delay, we can derive the worst-case frame forwarding time $R_i(q)$ of the q-th frame of a preemptable traffic stream i by evaluating all of its arrival candidates $a \in A_{q,i}$. However, as the queueing delay under frame preemption (cf. Definition 33) is defined to be the time until the last non-preemptable part (frame or fragment) can be sent, we have to consider this last part when deriving the worst-case frame forwarding time. For preemptable traffic streams with frames that are too small to be preepmted, i.e. $C_i^+ \leq \frac{143\,\text{bytes}}{r_{TX}}$, this part is the frame itself (cf. Theorem 15). For preemptable traffic streams with frames that can be preempted, i.e. $C_i^+ > \frac{143\,\text{bytes}}{r_{TX}}$, this part is the smallest fragment size (84 bytes, already including 24 bytes of preemption overhead), assuming that, in the worst-case, their remaining payload has been used during the computation of the queueing delay (as we did in Eq. (4.64)). □

The worst-case frame forwarding time (over all frame arrivals) of a preemptable traffic stream i can then be computed as in Theorem 2.

4.6.2.2 Resource-Level Analysis of IEEE 802.1Qbv with IEEE 802.3br

Originally, frame transmission in IEEE 802.1Qbv is non-preemptive. As discussed in Section 4.4, IEEE 802.1Qbv uses guard bands to protect the ST intervals from (less-critical) non-ST frames extending into these intervals and,

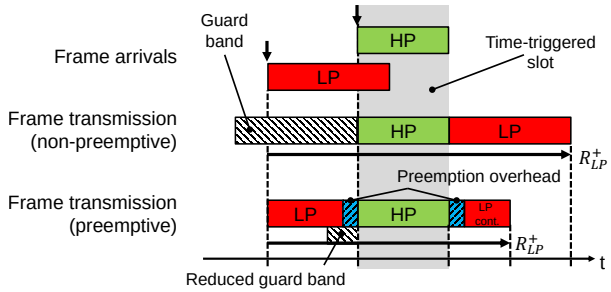


Figure 4.8: Example of non-preemptive and preemptive frame transmission in IEEE 802.1Qbv

hence, delaying the corresponding ST streams. During these guard bands, which must be of the length of the longest non-ST frame, no non-ST frame transmission is allowed to start. This can result in poor link utilization.

As the intention of preemption is to reduce the latency of critical traffic, we assume that ST streams are treated as express traffic, implying that ST frames cannot be preempted. As IEEE 802.1Qbv already guarantees their forwarding free from interference from other traffic classes as soon as their ST interval starts, they do not benefit from the preemption of other traffic. As proposed by [180], (less-critical) non-ST streams are assigned to the set of preemptable classes. Under frame preemption, the performance of ST streams can be improved, as now the guard bands can be reduced to the maximum non-preemptable fragment length (see Lemma 19).

This is illustrated in Figure 4.8. Without preemption, the non-ST frame *LP*, which arrives during its guard band, is blocked to prevent overlap with the ST interval of the *HP* frame. Only after *HP*'s ST interval is over, *LP* can transmit. With preemption, however, the guard band is smaller, so that in this particular example *LP* can transmit immediately after its arrival. *LP* can transmit until it is preempted by *HP*'s ST interval. It resumes transmission after the ST interval is over, leading to a shorter frame forwarding delay R^+_{LP} and better link utilization than in the non-preemptive scenario.

As discussed, ST streams, by design, do not benefit from frame preemption. Consequently, we only present a resource-level analysis for frame preemption according to IEEE 802.3br for non-ST streams in IEEE 802.1Qbv.

4.6.2.2.1 Resource-Level Analysis of Preemptable Traffic

Since frames of non-ST streams use the preemptable MAC interface of IEEE 802.1Qbv, they cannot preempt each other regardless of their priority. They can only be preempted by IEEE 802.1Qbv's scheduler, i.e. to insert ST intervals. Hence, lower-, same-, and higher-priority blocking can be modeled as in IEEE 802.1Qbv.

Lemma 32. *The lower-, same-, and higher-priority blocking for a preemptable non-ST stream in IEEE 802.3br for IEEE 802.1Qbv can be bounded as in Lemmata 24, 25, and 26, respectively.*

Proof. Follows by construction from Section 4.6.2.2. □

The additional periodic blocking by the ST intervals of ST classes can be bounded similar to Lemma 11. In IEEE 802.3br, however, we can exploit that frame preemption can reduce the guard band and, hence, the blocking.

Again, we start by bounding the maximum blocking a single ST interval can cause.

Lemma 33. *The maximum blocking caused by a single ST interval t_J^{ST} from ST class $J \in \mathcal{S}$ on preemptable non-ST classes in IEEE 802.3br for IEEE 802.1Qbv is bounded by*

$$\tilde{t}_J^{ST} = \min \left\{ \max_{i \in \bigcup\limits_{I \in \mathcal{P}} I} \left\{ C_i^+ \right\}, \frac{143\,\text{bytes}}{r_{TX}} \right\} + t_J^{ST} \tag{4.74}$$

where t_J^{ST} is the ST interval length of ST class J as defined in Definition 27.

Proof. The proof is similar to Lemma 10. By construction (cf. Section 4.6.2.2) all non-ST classes are preemptable classes. The minimum term models the minimum guard band length required to protect the ST interval of class J from overlapping frames of non-ST streams. This length is either the maximum non-preemptable fragment length $\frac{143\,\text{bytes}}{r_{TX}}$ (Lemma 19) or the maximum over all frames of non-ST streams, which becomes relevant if these frames are smaller than or equal to $\frac{143\,\text{bytes}}{r_{TX}}$. We do not know in which order the non-ST frames are processed. Thus, in the worst-case, we have to assume that non-ST streams arrive such that a frame or fragment of the length of the guard band becomes ready to transmit just after J's guard band started and that the guard band cannot be used to transmit any non-ST frames or fragments. The second term is the length of the time-triggered slot of ST class J. □

Now, we can bound the blocking by ST classes similar to Lemma 11.

Lemma 34 (Blocking by ST Classes in IEEE 802.3br for IEEE 802.1Qbv)**.** *In any time interval Δt, the maximum interference the frames of a preemptable non-ST stream i in IEEE 802.3br for IEEE 802.1Qbv can experience by a ST class J is bounded by*

$$I_i^{STB}(\Delta t) = \sum_{J \in \mathcal{S}} \left\lceil \frac{\Delta t}{t_J^{CYC}} \right\rceil \tilde{t}_{cl(i),J}^{ST} \tag{4.75}$$

where t_J^{CYC} is the ST cycle length for ST class J as defined in Definition 27.

Proof. Follows the proof of Lemma 11 with Lemma 33. □

Next, we derive an upper bound on the preemption overhead in IEEE 802.3br for IEEE 802.1Qbv.

Lemma 35 (Preemption Overhead for non-ST Streams in IEEE 802.3br for IEEE 802.1Qbv)**.** *The preemption overhead in a time interval of length Δt for a non-ST stream i in IEEE 802.3br for IEEE 802.1Qbv is upper bounded by*

$$I_i^{PO}(\Delta t) = \begin{cases} 0 & \text{if } \max\limits_{j \in \bigcup\limits_{J \in \mathcal{P}} J} \left\{C_j^+\right\} \leq \frac{143\,\text{bytes}}{r_{TX}} \\ \frac{24\,\text{bytes}}{r_{TX}} \sum\limits_{J \in \mathcal{S}} \left\lceil \frac{\Delta t}{t_J^{CYC}} \right\rceil & \text{otherwise} \end{cases} \tag{4.76}$$

Proof. If all non-ST frames are too small to be preempted, then there cannot be any preemption and hence there is no preemption overhead. Otherwise, as the non-ST streams cannot preempt themselves, there can only be one frame preemption per ST interval. In any time interval Δt, the maximum number of preemptions can be found by summing over the maximum number of ST intervals of all ST classes J in Δt (cf. Lemma (11)). Multiplying this number with the preemption overhead of 24 bytes (see Lemma 20) divided by r_{TX} yields the preemption overhead. □

With the help of the lower-, same-, and higher-priority bocking terms as well as the blocking from ST classes and the preemption overhead the multiple-event queueing delay can be bounded.

Theorem 44. *The multiple-event queueing delay in IEEE 802.3br for IEEE 802.1Qbv for the q-th frame arrival of preemptable traffic stream i that arrived at time a is upper bounded by*

$$Q_{i,a}(q) = I_i^{LPB} + I_i^{SPB}(q,a) + I_i^{HPB}(Q_{i,a}(q)) + I_i^{STB}(Q_{i,a}(q)) + I_i^{PO}(Q_{i,a}(q)) \tag{4.77}$$

Proof. The multiple-event queueing delay must take all blocking effects and the preemption overhead into account. From Lemmata 32, 34 and 35 the theorem follows (cf. proof of Theorem 18). □

With the multiple-event queueing delay, the multiple-event processing time can be bounded.

Theorem 45. *The multiple-event processing time for the q-th frame arrival of preemptable traffic stream i in IEEE 802.3br for IEEE 802.1Qbv is upper bounded by*

$$B_i^+(q) = Q_i(q) + C_i^+ \tag{4.78}$$

where $Q_i(q)$ is an upper bound on the multiple-event queueing delay for the q-th frame arrival of preemptable traffic stream i:

$$Q_i(q) = I_i^{LPB} + I_i^{SPB}(q, Q_i(q)) + I_i^{HPB}(Q_i(q)) + I_i^{STB}(Q_i(q)) + I_i^{PO}(Q_i(q)) \tag{4.79}$$

Proof. Follows the proofs of Theorems 9 and 12 (cf. proof of Theorem 19) with Theorem 44. □

With the help of Lemma 32 as well as the preemption overhead from Lemma 35, the multiple-event scheduling horizon can be derived.

Theorem 46. *The multiple-event scheduling horizon for the q-th frame arrival of a preemptable non-ST stream i in IEEE 802.3br for IEEE 802.1Qbv is upper bounded by*

$$S_i(q) = I_i^{LPB} + \tilde{I}_i^{SPB}(q, S_i(q)) + \tilde{I}_i^{HPB}(S_i(q)) + I_i^{STB}(S_i(q)) + I_i^{PO}(q, S_i(q)) \quad (4.80)$$

with

$$\tilde{I}_i^{SPB}(q, \Delta t) = qC_i^+ + \sum_{j \in fifo(i)} \eta_j^+(\Delta t) C_j^+ \quad (4.81)$$

and

$$\tilde{I}_i^{HPB}(\Delta t) = \sum_{j \in hp(i)} \eta_j^+(\Delta t) C_j^+ \quad (4.82)$$

Proof. Follows the proof of Theorems 20 and 42. □

The worst-case frame forwarding times and maximum buffer size requirements for preemptable non-ST traffic streams in IEEE 802.3br for IEEE 802.1Qbv can be computed as described in Theorem 43 and Section 4.3.4, respectively.

4.7 Considering Workload Correlations on Shared Output Ports

Typically, CPA conservatively assumes that all event models are independent of each other (cf. Section 3.3.2) and that there is no limit on the amount of workload that can arrive instantaneously at resources. While this might be a valid assumption for the performance analysis on processors, where tasks and their respective load (execution times) may be released with potential infinitesimal distance between them, frames in an actual Ethernet network cannot be transmitted faster than the limited transmission rate r_{TX} of the output port of the switches, e.g. 100 MBit/s or 1 GBit/s. Consequently, output ports act as (natural) traffic shapers that limit the amount of workload that can pass them in a certain amount of time.

The exploitation of workload correlation we are presenting in this thesis is a generalization of [17] and [167]. We extend the work started in these publications to support heterogeneous networks with different link speeds, as they are common in the automotive context, and also consider the correlation of different correlated traffic classes that share a link. This form of workload correlation

can also be found in other formal analysis approaches in the Ethernet context such as [53], [24], and [60].

In the context of Ethernet, we define workload as follows.

Definition 34 (Workload)**.** *The workload that a traffic stream releases in a given time interval Δt is the amount of time required to transmit all frames of this traffic stream that arrive during Δt.*

Typically, the amount of workload a traffic stream i releases in a given time interval Δt is upper bounded by evaluating the traffic stream's upper frame arrival function over Δt and multiply the result with the stream's maximum frame transmission time C_i^+. Note that the maximum workload released by a single frame is time-independent and corresponds to the frame's maximum frame transmission time. Definition 34 can be extended to a set of traffic streams by summation of the workloads of all individual traffic streams in this set.

As a single output port imposes a limit on the combined workload of all the streams passing it, we say that these streams are workload correlated.

In the following, we will see that the correlated workload limit depends on the transmission rates of the currently analyzed port, i.e. CPA resource, and its predecessor ports. To aid the discussion, we introduce the following definitions.

Definition 35 (Current Output Port)**.** *Let $res(i)$ be the output port that we are currently considering as part of the analysis of the traffic stream i.*

Definition 36 (Immediate Predecessor Output Port)**.** *Let $pre_J(r)$ and $pre_j(r)$ be the output port that is the immediate predecessor port of output port r along the path of the traffic streams in set J and along the path of the traffic stream j, respectively. If the traffic streams in J do not have the same common predecessor or no predecessor at all, then $pre_J(r)$ is undefined.*

Definition 37 (Port Transmission Rate of a Specific Port)**.** *Let r_{TX}^h be the port transmission rate of output port h.*

For instance, if the currently analyzed port, i.e. the one with the currently analyzed traffic stream $i \in I$ mapped to it, is $res(i)$, then the transmission rate of its immediate predecessor output port is $r_{TX}^{pre_I(res(i))}$.

Next, we define two sets that help us to efficiently reason about correlated traffic streams that use a common output port.

Definition 38 (Traffic Streams with Common Predecessors)**.** *We consider all the traffic streams that share the currently analyzed port with traffic stream i. Let Γ_i be a set of sets. On the first level, these sets group the traffic streams by their immediate predecessor output port, i.e. there is one set per common predecessor port. Each of these sets, in turn, contains the traffic streams from the set's respective predecessor port.*

Traffic streams that do not have a predecessor port are not considered in the subsets of Γ_i. Examples of traffic streams on the currently analyzed port that do not have an immediate predecessor port are traffic streams that are

triggered by environment event models or traffic streams that arrive from non-Ethernet resources.

Definition 39 (Traffic Streams without Common Predecessors)**.** *Out of all the traffic streams, that share the currently analyzed port with traffic stream i, let $\bar{\Gamma}_i$ be the set of traffic streams without any predecessor output port.*

Note that, in particular, we have that $\bigcup_{I \in \Gamma_i} I \cup \bar{\Gamma}_i$ is the set of all traffic streams on $res(i)$ with $\forall I, J \in \Gamma_i : I \cap J = \emptyset$ and $\forall I \in \Gamma_i : I \cap \bar{\Gamma}_i = \emptyset$.

Now, we can start bounding the workload that can be released by correlated traffic streams that share an output port.

Theorem 47 (Correlated Workload Limit)**.** *The workload that is released in a time interval Δt by a set of correlated traffic streams J with an uncorrelated workload of Δw can be bounded by*

$$H^+_{i,J}(\Delta w, \Delta t) = \min \left\{ \Delta w, \underbrace{\frac{r_{TX}^{pre_J(res(i))}}{r_{TX}^{res(i)}} \Delta t}_{\text{(a)}} + \underbrace{\max_{j \in J} \left\{ C_j^+ \right\}}_{\text{(b)}} \right\} \tag{4.83}$$

$H^+_{i,J}(\Delta t)$ is undefined if the traffic streams in J do not have the same common predecessor or no predecessor at all.

Proof. First term of the minimum: This is by definition the actual uncorrelated workload.

Second term of the minimum: This is a workload limit for the traffic streams in set J that is independent from the actual frame arrival patterns. In the worst-case, a frame releases its workload as early as possible, i.e. at the instance of its arrival. As Ethernet ports transmit frames, the time it takes to transmit a frame corresponds to its workload. The predecessor port $pre_J(res(i))$ that is shared by the correlated traffic streams in J defines how fast frames (and hence workload) can arrive from this predecessor at the currently analyzed port $res(i)$. In the worst-case, the largest frame among the traffic streams J arrives first and releases its workload (term (b)). Here, we assume that C_j^+ in term (b) already is the workload from the perspective of the transmission rate of the currently analyzed port. Term (a) models that, after this frame has arrived, the arrival of any additional workload is limited by the time interval during which this workload arrives, i.e. in any time interval Δt only Δt worth of workload can arrive. To compensate for different port transmission rates between the predecessor and the current port, we introduce a scaling factor of $(r_{TX}^{pre_J(res(i))} / r_{TX}^{res(i)})$.

As the uncorrelated workload might be smaller than the arrival-independent workload bound, we take the minimum of both terms. □

Note that Theorem 47 assumes that Δt is continuous. The FIFO candidate search that is part of the Ethernet analysis, however, selects its candidates at points where the workload increases (cf. Theorem 16), resulting in an unbounded number of candidates. Hence, it makes sense to discretize the correlated workload limit as proposed in [17].

Theorem 48 (Discretized Correlated Workload Limit)**.** *The discretized workload that is released in a time interval Δt by a set of correlated traffic streams J with an uncorrelated workload of Δw can be bounded by*

$$H^+_{i,J,s}(\Delta w, \Delta t) = \min \left\{ \Delta w,\ \frac{r_{TX}^{pre_J(res(i))}}{r_{TX}^{res(i)}} \left\lceil \frac{\Delta t}{s} \right\rceil s + \max_{j \in J} \left\{ C_j^+ \right\} \right\} \tag{4.84}$$

where s is the step size via which $(r_{TX}^{pre_J(res(i))}/r_{TX}^{res(i)})\Delta t + \max_{j \in J}\{C_j^+\}$ is discretized.

Proof. Follows from Theorem 47 and that $\forall s : (r_{TX}^{pre_J(res(i))}/r_{TX}^{res(i)})\lceil \Delta t/s \rceil s + \max_{j \in J}\{C_j^+\} \geq (r_{TX}^{pre_J(res(i))}/r_{TX}^{res(i)})\Delta t + \max_{j \in J}\{C_j^+\}$. □

The discretized correlated workload limit can be used to derive new (potentially tighter) bounds on the multiple-event queueing delay, the multiple-event processing time, and the multiple-event scheduling horizon. We proceed to derive these bounds in the context of IEEE 802.1Q. Other transmission selection mechanisms can be bounded similarly, while keeping in mind that only blocking terms that are a result of frames being transmitted can potentially exploit workload correlations.

4.7.1 Exploiting Workload Correlations in IEEE 802.1Q

In this section we explain how the resource-level analysis of IEEE 802.1Q can be extended to exploit workload correlations between traffic streams that share resources. First, we redefine the individual blocking terms of IEEE 802.1Q to support correlated traffic streams.

We can reuse the lower-priority blocking without modification from IEEE 802.1Q.

Lemma 36 (Lower-Priority Blocking in IEEE 802.1Q with Workload Correlation)**.** *The lower-priority blocking a traffic stream i can experience in IEEE 802.1Q while exploiting workload correlation on shared output ports is bounded as for IEEE 802.1Q in Eq. (4.3).*

Proof. As there is only one lower priority blocker, we cannot exploit any correlation. Hence the lemma follows directly from Lemma 1. □

The same- and higher-priority blocking terms are each split into two terms: one to compute the interference from tasks without any predecessor output port (i.e. uncorrelated tasks) and one to compute the interference from individual sets of correlated tasks.

Lemma 37 (Same-Priority Blocking in IEEE 802.1Q with Workload Correlation)**.** *The same-priority blocking the q-th frame of a traffic stream i that arrived at time a can experience in IEEE 802.1Q from its own frames and frames from*

interfering traffic streams (in $\bar{\Gamma}_i$) that do not have a predecessor output port is bounded by

$$\bar{I}_i^{SPB}(q,a) = (q-1)C_i^+ + \sum_{j \in uncorfifo(i)} \eta_j^{+]}(a) C_j^+ \tag{4.85}$$

and the same-priority blocking the q-th frame of a traffic stream i that arrived at time a can experience in IEEE 802.1Q from the frames of a set of correlated traffic streams J, i.e. traffic streams that share an immediate common predecessor output port, is bounded by

$$I_{i,J,s}^{SPB}(a) = H_{i,J,s}^+ \left(\sum_{j \in J} \eta_j^{+]}(a) C_j^+, a \right) \tag{4.86}$$

where $uncorfifo(i) = \bar{\Gamma}_i \cap fifo(i)$ is the set of interfering traffic streams that do not have a predecessor output port and interfere with task i in a FIFO fashion (cf. Thorem 11).

An upper bound on the entire same-priority blocking can be derived by summing Eq. (4.86) for all sets in Γ_i and adding the result to Eq. (4.85).

$$\bar{I}_i^{SPB}(q,a) + \sum_{J \in \Gamma_i} I_{i,corfifo(i,J),s}^{SPB}(a) \tag{4.87}$$

where $corfifo(i,J) = J \cap fifo(i)$ is the set of interfering traffic streams that share a common predecessor output port and interfere with task i in a FIFO fashion.

Proof. Follows the proof of Lemma 2, with the exception that, here, we split the interference in traffic streams without any predecessor output port (Eq. (4.85)) and traffic streams with a common predecessor output port (Eq. (4.86)). As the interference from the latter is correlated by construction, we can apply the discretized correlated workload limit from Theorem 48. □

Lemma 38 (Higher-Priority Blocking in IEEE 802.1Q with Workload Correlation)**.** *The higher-priority blocking a traffic stream i can experience in any time interval Δt in IEEE 802.1Q from the frames of interfering traffic streams (in $\bar{\Gamma}_i$) that do not have a predecessor output port is bounded by*

$$\bar{I}_i^{HPB}(\Delta t) = \sum_{j \in uncorhp(i)} \eta_j^{+]}(\Delta t) C_j^+ \tag{4.88}$$

and the higher-priority blocking a traffic stream i can experience in any time interval Δt in IEEE 802.1Q from the frames of a set of correlated traffic streams J, i.e. traffic streams that share an immediate common predecessor output port, is bounded by

$$I_{i,J,s}^{HPB}(\Delta t) = H_{i,J,s}^+ \left(\sum_{j \in J} \eta_j^{+]}(\Delta t) C_j^+, \Delta t \right) \tag{4.89}$$

where $uncorhp(i) = \bar{\Gamma}_i \cap hp(i)$ *is the set of interfering traffic streams that do not have a predecessor output port and that have a priority higher than that of stream* i.

An upper bound on the entire higher-priority blocking can be derived by summing Eq. (4.89) for all sets in Γ_i *and adding the result to Eq. (4.88).*

$$\bar{I}_i^{HPB}(\Delta t) + \sum_{J \in \Gamma_i} I_{i,corhp(i,J),s}^{HPB}(\Delta t) \tag{4.90}$$

where $corhp(i, J) = J \cap hp(i)$ *is the set of interfering traffic streams that share a common predecessor output port and have a priority higher than that of stream* i.

Proof. Follows the proof of Lemma 3, with the exception that, here, we split the interference in traffic streams without any predecessor output port (Eq. (4.88)) and traffic streams with a common predecessor output port (Eq. (4.89)). As the interference from the latter is correlated by construction, we can apply the discretized correlated workload limit from Theorem 48. □

Now, we can refine the multiple-event queueing delay, the multiple-event processing time, and the multiple-event scheduling horizon of IEEE 802.1Q to exploit potential workload correlation on shared output ports.

Theorem 49. *The multiple-event queueing delay in IEEE 802.1Q with workload correlation for the* q*-th frame arrival of traffic stream* i *that arrived at time* a *is upper bounded by*

$$\begin{aligned} Q_{i,a,s}(q) = I_i^{LPB} + \bar{I}_i^{SPB}(q,a) + \bar{I}_i^{HPB}(Q_{i,a,s}(q)) + \\ \sum_{J \in \Gamma_i} H^+_{i,corfifo(i,J) \cup corhp(i,J),s} \Big(I^{SPB}_{i,corfifo(i,J),s}(a) + \\ I^{HPB}_{i,corhp(i,J),s}(Q_{i,a,s}(q)), Q_{i,a,s}(q) \Big) \end{aligned} \tag{4.91}$$

where s *is the step size of the discretized correlated workload limit.*

Proof. Follows the proof of Theorem 18 with Lemmata 36, 37, and 38. Additionally, we can apply the discretized correlated workload limit from Theorem 48 to limit the combined interference from same- and higher-priority traffic stream that share a common predecessor output port. □

The multiple-event processing time and the multiple-event scheduling horizon can be bounded similarly.

Theorem 50. *The multiple-event processing time for the* q*-th frame arrival of traffic stream* i *in IEEE 802.1Q with workload correlation is upper bounded by*

$$B^+_{i,s}(q) = Q_{i,s}(q) + C_i^+ \tag{4.92}$$

where s is the step size of the discretized correlated workload limit and $Q_{i,s}(q)$ is an upper bound on the multiple-event queueing delay for the q-th frame arrival of traffic stream i in IEEE 802.1Q with workload correlation:

$$Q_{i,s}(q) = I_i^{LPB} + \bar{I}_i^{SPB}(q, Q_{i,s}(q)) + \bar{I}_i^{HPB}(Q_{i,s}(q)) + \sum_{J \in \Gamma_i} H^+_{i,corfifo(i,J) \cup corhp(i,J),s} \Big(I^{SPB}_{i,corfifo(i,J),s}(Q_{i,s}(q)) + I^{HPB}_{i,corhp(i,J),s}(Q_{i,s}(q)), Q_{i,s}(q) \Big) \quad (4.93)$$

Proof. Follows the proofs of Theorem 19 and Theorem 48 with the argumentation from Theorem 49. □

Theorem 51. *The multiple-event scheduling horizon for the q-th frame arrival of traffic stream i in IEEE 802.1Q is upper bounded by*

$$S_{i,s}(q) = I_i^{LPB} + \tilde{\bar{I}}_i^{SPB}(q, S_{i,s}(q)) + \tilde{\bar{I}}_i^{HPB}(S_{i,s}(q)) + \sum_{J \in \Gamma_i} H^+_{i,corfifo(i,J) \cup corhp(i,J),s} \Big(\tilde{I}^{SPB}_{i,corfifo(i,J),s}(S_{i,s}(q)) + \tilde{I}^{HPB}_{i,corhp(i,J),s}(S_{i,s}(q)), S_{i,s}(q) \Big) \quad (4.94)$$

with

$$\tilde{\bar{I}}_i^{SPB}(q, \Delta t) = qC_i^+ + \sum_{j \in uncorfifo(i)} \eta_j^+(\Delta t) C_j^+ \quad (4.95)$$

and

$$\tilde{I}^{SPB}_{i,J,s}(\Delta t) = H^+_{i,J,s} \left(\sum_{j \in J} \eta_j^+(\Delta t) C_j^+, \Delta t \right) \quad (4.96)$$

and

$$\tilde{\bar{I}}_i^{HPB}(\Delta t) = \sum_{j \in uncorhp(i)} \eta_j^+(\Delta t) C_j^+ \quad (4.97)$$

and

$$\tilde{I}^{HPB}_{i,J,s}(\Delta t) = H^+_{i,J,s} \left(\sum_{j \in J} \eta_j^+(\Delta t) C_j^+, \Delta t \right) \quad (4.98)$$

where s is the step size of the discretized correlated workload limit.

Proof. Follows the proofs of Theorem 20 and Theorem 48 with the argumentation from Theorem 49. □

The worst-case frame forwarding times and maximum buffer size requirements can be computed as described in Section 4.3.2 and Section 4.3.4. However, the set of candidates must be extended to include the times where the discretized correlated workload is incremented. The set of arrival times for the q-th frame arrival of a traffic stream i with workload correlation can be constrained as follows [17].

Theorem 52. *The set $A_{q,i}$ of arrival times to be evaluated for the q-th frame arrival of a traffic stream i under workload correlation can be limited to the times where a coincides with the arrival times of interfering task activations or the steps of the discretized correlated workload limit:*

$$A_{q,i,s} = \bigcup_{j \in fifo(i)} \left\{ \delta_j^-(n) \,|\, \delta_i^-(q) \leq \delta_j^-(n) < S_{i,s}(q) \right\}_{n \in \mathbb{N}^{>0}} \cup \left\{ ns \,|\, \delta_i^-(q) \leq ns < S_{i,s}(q) \right\}_{n \in \mathbb{N}^{>0}} \quad (4.99)$$

where s is the step size of the discretized correlated workload limit.

Proof. Follows the proof of Theorem 16 and the fact that the steps (workload increments) of the discretized correlated workload limit are at ns. □

As can be seen, a smaller step size leads to more candidates. From Eq. (4.84) we see that a smaller step size also results in a higher accuracy (the result is always a conservative upper bound). Hence, there is a trade-off between accuracy and computational effort. Setting the step size s to one bit time, i.e. $1/r_{TX}^{res(i)}$, yields the highest accuracy.

4.7.2 Exploiting Workload Correlations in other Transmission Selection Mechanisms

While the (discretized) correlated workload limit, as a physical limitation, is a valid upper bound for all transmission selection mechanisms, it can often be improved by exploiting intrinsic properties of a given transmission selection mechanism.

For Ethernet AVB's transmission selection mechanism, the additional bandwidth limitation from the credit-based shaping algorithm can be exploited [17].

Tighter bounds for IEEE 802.1Qbv can by derived by exploiting the schedule of the ST intervals defined by its gate control list. This schedule is fixed and known at design time. Since ST streams can only transmit frames inside the ST intervals of their traffic class and, conversely, non-ST streams can only transmit frames when no ST intervals are scheduled, the ST interval schedule imposes an additional bandwidth limitation (assuming that guard bands can be fully utilized). This optimization has been implemented in the IEEE 802.1Qbv analysis presented in this thesis.

4.8 Summary

In this chapter, we presented how the performance analysis of Ethernet networks can be embedded into the CPA approach from Chapter 3. We introduced an Ethernet-specific system model and showed how it maps to the CPA system model. Using this Ethernet system model, we provided resource-level analyses to evaluate the worst-case performance guarantees of different Ethernet transmission selection mechanisms, focusing on the ones that are standardized as part of Ethernet TSN: static priority based transmission selection (IEEE 802.1Q), scheduled traffic (IEEE 802.1Qbv), cyclic queueing and forwarding (IEEE 802.1Qch), and frame preemption (IEEE 802.3br) for both IEEE 802.1Q and IEEE 802.1Qbv. These guarantees include bounds on the end-to-end path latencies and switch buffer sizes, which are of particular interest in automotive Ethernet networks. Furthermore, we showed how the intrinsic traffic shaping properties of Ethernet links can be exploited by considering the correlation of traffic streams that jointly pass a set of switch output ports.

Additionally, we formalized design considerations for IEEE 802.1Qbv and IEEE 802.1Qch, stating under which conditions these standards perform as intended. For IEEE 802.1Qch, in particular, we argued that, while it does achieve to simplify the latency computation, one needs a formal analysis (such as the one presented in this chapter) to ensure that the preconditions for the simplified latency computation hold for non-trivial cases.

The presented analyses are fully compatible with the CPA approach. This allows the seamless integration with other CPAs, e.g. to analyze entire cause effect chains extending over multiple (different) buses and CPUs, e.g. [172].

In the next chapter we evaluate the worst-case guarantees from the presented transmission selection mechanisms.

Chapter 5

Evaluation

In this chapter, we evaluate the timing analyses that have been proposed in this thesis. In particular, we will (1) in Section 5.2, quantify the effect of the optimization from Section 4.7 by comparing the results from this optimization to the results from the baseline IEEE 802.1Q timing analysis, (2) in Section 5.3, evaluate the worst-case performance of Ethernet TSN's IEEE 802.1Qbv and IEEE 802.1Qch transmission selection mechanisms and compare it to the worst-case performance of IEEE 802.1Q, and (3) in Section 5.4, quantify the benefit of IEEE 802.3br frame preemption for IEEE 802.1Q and IEEE 802.1Qbv under worst-case conditions. Throughout our evaluation, We use the worst-case end-to-end latency guarantee from our analyses as the comparison metric.

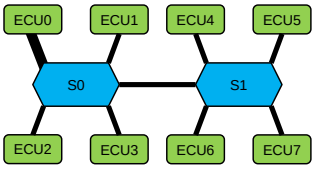
5.1 Network Setup

5.1.1 Network Topologies

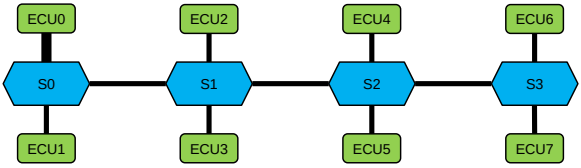
In our evaluation, we focus on the network topologies shown in Figure 5.1, namely the double star, quad star, tree, and line topologies. These topologies have been chosen to model (anticipated) automotive Ethernet topologies [67] [68]. Each topology is defined by the number and interconnection of its switches. For each topology, a fixed number of eight ECUs (ECU0 to ECU7) is distributed across the network. Highly-loaded links are 1 GBit/s (e.g. 1000 BASE-T1 [2]), while all other links are 100 MBit/s (e.g. BroadR-Reach [29] or 100 BASE-T1 [4]).

5.1.2 Network Traffic

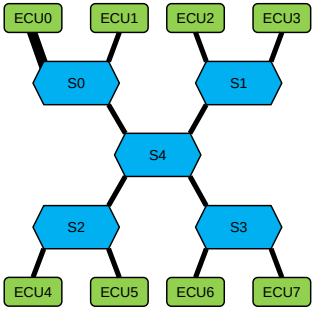
In this section we motivate and describe the network traffic used during our evaluation. We classify the traffic into three classes with different timing and bandwidth requirements: control traffic, camera traffic, and background traffic. In the following, we provide a short summary of the traffic characteristics of each class. Details can be found in Table A.1 in Appendix A. Altogether, there



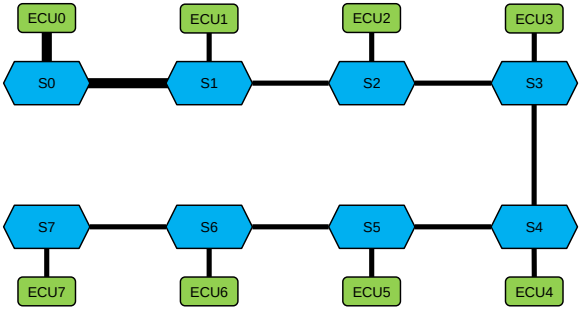
(a) Double star



(b) Quad star



(c) Tree



(d) Line

Figure 5.1: Topologies investigated during evaluation. Normal lines indicate 100 MBit/s links, bold lines indicate 1 GBit/s links.

are 451 individual traffic paths in the network (if each path of a multicast and broadcast tree is counted separately).

Note that the payload in Table A.1 represents the payload of each frame of a traffic stream. Additionally, we use IPv4 on the network layer and UDP on the transport layer, which account for an additional overhead of 28 bytes (20 bytes for IPv4 and 8 bytes for UDP, cf. Section 2.4). This overhead must be considered when computing the actual Ethernet frame transmission time bounds via Eqs. (4.1) and (4.2).

Also note that, we use a periodic with jitter event model (cf. Section 3.2.2.2) where the jitter equals the period to model that, in the worst-case, there might be an occasional burst of two frames entering the network.

5.1.2.1 Control Traffic

Control traffic typically comprises latency-critical traffic streams with low bandwidth requirements. In our setup, we divide it into unicast, multicast, and broadcast traffic.

Unicast traffic is used for the direct communication between two communication partners. Here, unicast traffic streams use periods of 5, 10, 20, 50, 100 and 200 ms with payloads of 8, 64, 96, 128, 160, 192, and 256 bytes.

Multicast traffic is typically used to implement publisher/subscriber communication, such as SOME/IP notifications, and can help to use the network resources more efficiently. Another important use-case of multicast communication is SOME/IP service discovery [37]. In our setup, multicast traffic streams use periods of 5, 50, and 200 ms with payloads of 8, 64, and 256 bytes.

Broadcast traffic is used to reach all end stations connected to a network. Examples of broadcast traffic include periodic ECU status messages. In this evaluation, broadcast traffic streams use periods of 20, 50, and 100 ms with payloads of 32, 64, and 128 bytes.

The accumulated bandwidth of the control traffic injected by each ECU is about 2.25 MBit/s. In our evaluation, we map control traffic to PCP 3 (and additionally to PCP 4 for selected traffic streams in Sections 5.3 and 5.4), which is the highest priority.

5.1.2.2 Camera Traffic

Camera traffic is typically high-bandwidth traffic. It can also be latency critical, e.g. if the traffic contributes to a safety function.

There are four unicast camera traffic streams (e.g. to implement a surround view system) in the network, which are sent from ECU1, ECU3, ECU5, and ECU7 to ECU0 (e.g. for data fusion and/or display on a head unit). The bandwidth requirement for each camera stream is 25.12 MBit/s (cf. [18]). In our evaluation, we map camera traffic to PCP 2, which is lower than the priority level of control traffic, but higher that the priority level of background traffic.

5.1.2.3 Background Traffic

In order to stress the network, we introduce low-priority, non-latency-critical background traffic into the network. This is a typical approach in network

performance evaluation, e.g. [127]. In our setup, the main motivation of this background traffic is to emulate lower-priority blocking on each switch port. This is important since, as we discussed in Chapter 4, even lower-priority traffic can have a (negative) impact on the timing guarantees for higher-priority traffic in most Ethernet transmission selection mechanisms.

This background traffic is modeled by traffic streams with maximum frame size that are broadcasted as low-priority interference on PCP 1 (i.e. the lowest priority), originating from ECU0 and ECU7 in the double star, quad star, and line topologies and from ECU0, ECU2, ECU4, and ECU7 for the tree topology.

All streams have a period of 1 s. As this traffic is, by definition, not latency-critical, we will neither derive worst-case latency guarantees nor discuss the performance of this traffic.

5.1.3 Presentation of Results

Since we are interested in the overall performance of different transmission selection mechanisms, it is not productive to compare 451 paths individually. Instead, we reduce these individual results into an easily comprehensible form.

In the following discussion, we mainly use box plots [124] to summarize the worst-case end-to-end latency guarantees of all streams of a given traffic class (often with a certain set of parameters). Note that each value contributing to the box plot represents a worst-case end-to-end latency guarantee derived by the formal analysis method presented in this thesis. For each traffic class, the gray box covers 50 % of the worst-case end-to-end latency guarantees, with its lower and upper borders giving the 25 % and 75 % quartiles, respectively. The whiskers indicate the worst-case guarantees of the streams with the shortest and longest latency guarantees (among all paths). The median and average among the worst-case latency guarantees are marked by a red bar and a black dot, respectively. The x-axis labels identify the individual experiments.

5.2 IEEE 802.1Q: Baseline vs. Optimization

In this section, we compare the baseline timing analysis for IEEE 802.1Q from Section 4.3 to the optimization from Section 4.7 in order to quantify the general improvement focusing on the worst-case latency guarantees.

5.2.1 Experimental Setup

The experimental setup follows the description from Section 5.1.2. In particular, we use the priority mapping summarized in Table 5.1.

	Control Traffic	Camera Traffic	Background Traffic
PCP	3	2	1

Table 5.1: Priority code points of the evaluated traffic classes.

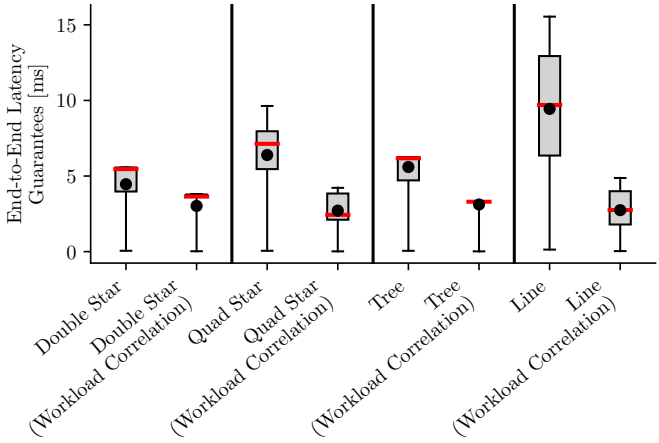


Figure 5.2: End-to-end latency guarantee comparison between the IEEE 802.1Q baseline analysis and the IEEE 802.1Q analysis with workload correlation of PCP 3 traffic

We will conduct our evaluation for the four different network topologies from Figure 5.1. This is important, as the topology (along with the network traffic) determines the exploitable correlation potential (cf. Definition 38). A star topology, for example, typically exhibits less potential to exploit correlations than a line topology.

5.2.2 Results

Figures 5.2 and 5.3 show the worst-case end-to-end latency guarantees for the plain analysis and the one exploiting workload correlations for each topology and priority as boxplots. Additionally, Table 5.2 shows the peak worst-case latency reduction for each setup in percent, i.e. how much the largest worst-case latencies in Figures 5.2 and 5.3 could be improved by applying the workload correlation.

As we can see, both control and camera traffic benefit from the optimized analysis. The two key observations are (1) higher-priority traffic streams (PCP 3) benefit more from our proposed optimization than lower-priority traffic streams (PCP 2) and (2) for each PCP, the degree of latency reduction through our optimization highly depends on the topology. We will discuss these observations further in the following subsections.

5.2.2.1 Topology Impact

As we see, the double star topology exhibits the least improvement. The quad star and tree topologies show (roughly) comparable improvements, while the line topology benefits the most from exploiting workload correlations.

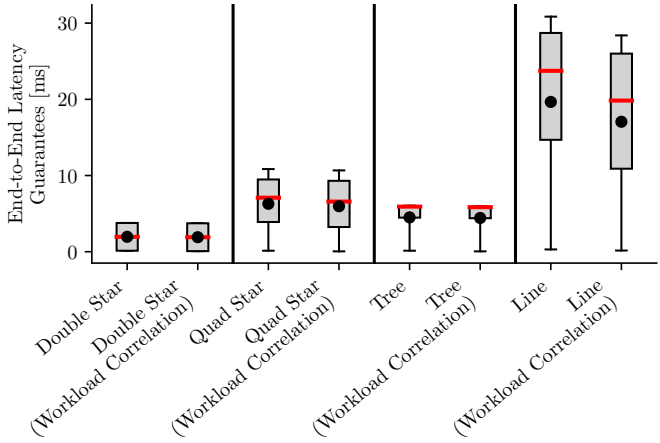


Figure 5.3: End-to-end latency guarantee comparison between the IEEE 802.1Q baseline analysis and the IEEE 802.1Q analysis with workload correlation of PCP 2 traffic

	Double Star	**Quad Star**	**Tree**	**Line**
PCP 3	31.56 %	56.20 %	46.51 %	68.66 %
PCP 2	1.08 %	1.63 %	1.16 %	7.99 %

Table 5.2: Peak improvements while exploiting workload correlation

As discussed, correlations only exists if two or more traffic streams share the same path in the network along at least two successive switch ports. In the double star topology this only applies to traffic streams that are sent across both switches and, on the second switch, use the same output port. Hence, the optimization potential is comparatively low. In the other topologies, the paths of the traffic streams are longer and, hence, the likelihood of traffic streams sharing resources is higher. Consequently, the optimization potential increases.

Note that, in compositional performance analysis, with each (shared) resource the propagated event models become more and more general, i.e. their upper and lower bounds drift further apart. This implies that the burst size or at least the potential for bursty behavior increases, which leads to more conservative analysis results such as longer worst-case latency guarantees. Exploiting workload correlations can help to reduce this effect, by reducing the worst-case response times during the resource-level analysis, which, in turn, leads to tighter output event models (cf. Theorem 5).

5.2.2.2 Priority Impact

For each topology, the higher-priority control traffic streams (PCP 3) show significantly larger latency improvements than the lower-priority (PCP 2) video traffic streams.

Control traffic is low bandwidth traffic (cf. Section 5.1.2.1), but there are many control traffic streams (240 in total) in the network. During resource-level analysis, CPA conservatively assumes that all traffic streams arrive concurrently. This can lead to large initial interference, which, in turn, can lead to large worst-case response times. When exploiting workload correlations, the interference is (per predecessor) limited to the link speed. More specifically, the amount of interference is limited by the link speed and the time interval during which this interference can occur (cf. Eq. (4.84)).

For control traffic, the main source of interference is same-priority control traffic. Same-priority blocking with workload correlation does not directly depend on the size of the investigated time interval (cf. Eq. (4.86)). Instead, it depends on a which remains constant during the fixed point iteration of the resource-level analysis. As a consequence, the interference from correlated traffic streams is upper bounded independent of the size of the (growing) multiple-event queuing delay. This can significantly reduce the impact of the large initial interference that CPA is susceptible of and, in our setup, leads to significant improvements.

For video traffic the situation is different. Here, we only have a total of 4 traffic streams. Consequently, there is significantly less potential to exploit workload correlations as part of same-priority blocking. Instead, there is higher-priority interference from control traffic, which could potentially be exploited. However, for higher-priority blocking with workload correlation the optimization potential is limited. Here, the discretized correlated workload limit depends on Δt and grows monotonically with it (cf. Eq. (4.89)). The amount of growth is determined by the scaling factor $\alpha = r_{TX}^{pre_J(res(i))}/r_{TX}^{res(i)}$, which depends on the transmission rates of the currently analyzed port and its predecessor. If $\alpha \geq 1$ (no change of transmission rates or the predecessor port has a higher transmission rate than the currently analyzed port), then the workload limit will grow with each iteration until Δw in Eq. (4.84) dominates. Hence, in this case, there is no benefit from applying the workload limit. However, if $\alpha < 1$ (the predecessor port has a lower transmission rate than the currently analyzed port), then there is a chance that Δt-dependent term in Eq. (4.84) dominates. In our setup, there is just one link speed change (from 100 MBit/s to 1 GBit/s) and, hence, there is only marginal improvement video traffic streams.

5.3 IEEE 802.1Q vs. IEEE 802.1Qbv vs. IEEE 802.1Qch

In this section, we focus on the evaluation of the worst-case performance of Ethernet TSN's time-aware and peristaltic shapers, IEEE 802.1Qbv and IEEE 802.1Qch respectively. In particular, we will apply these traffic shapers to a selected subset of high-priority latency-critical traffic to investigate the performance impact of Ethernet TSN's traffic shapers. Furthermore, we will take

a look at the timing impact of the shaped IEEE 802.1Qbv and IEEE 802.1Qch traffic classes on unshaped lower priority traffic (which might still be latency critical).

Additionally, we will compare the worst-case performance of IEEE 802.1Qbv and IEEE 802.1Qch to the worst-case performance of IEEE 802.1Q.

5.3.1 Experimental Setup

The experimental setup principally follows the description from Section 5.1.2. We introduce, however, an additional traffic class for latency-critical control traffic that will make use of the traffic shapers of IEEE 802.1Qbv and IEEE 802.1Qch. Traffic of this class will be called control data traffic (CDT) and is mapped to PCP 4 (i.e. the highest priority in our setup). For this evaluation, we map all traffic streams with a period of 5 ms (cf. Appendix A) to the CDT class.

We keep the remaining control traffic on PCP 3 and will refer to it as general control traffic (GCT). Camera and non-real-time background traffic remain as described in Sections 5.1.2.2 and 5.1.2.3 and will be referred to as CAM and NRT, respectively. GCT, CAM, and NRT traffic is unshaped, i.e. does not use the traffic shapers of IEEE 802.1Qbv or IEEE 802.1Qch and is scheduled based on its priority. Table 5.3 summarizes the mapping of traffic classes to priorities.

	Control Traffic		**Camera Traffic**	**Background Traffic**
	CDT	**GCT**	**CAM**	**NRT**
PCP	4	3	2	1

Table 5.3: Priority code points of the evaluate traffic classes.

For all analyses, we use the optimization proposed in Section 4.7 (including the proposal for IEEE 802.1Qbv from Section 4.7.2).

Next, we discuss how the traffic shapers of IEEE 802.1Qbv and IEEE 802.1Qch are configured for our evaluation.

5.3.1.1 Shaper Configurations of IEEE 802.1Qbv

For IEEE 802.1Qbv, we investigate three shaper configurations characterized by their ST interval length and their ST cycle time: *IEEE 802.1Qbv 450 us / 5 ms*, *IEEE 802.1Qbv 900 us / 5 ms*, and *IEEE 802.1Qbv 450 us / 5 ms sync*. The first two configurations assume, that, in the worst-case, the switches in the network are unsynchronized, i.e. subsequent ST intervals for the same ST class are not necessarily aligned (e.g. due to lost synchronization, misconfiguration, or an attack). The third configuration assumes that all switches in the network are synchronized and subsequent ST intervals for the same ST class are aligned, i.e. we assume that the ST intervals at each switch port are scheduled such that CDT traffic does not need to wait for its ST intervals and that all ST intervals are large enough to process any CDT traffic arriving within the interval. Each configuration is characterized by two time intervals. The first time interval

(450 us or 900 us) corresponds to the ST interval length, i.e. the interval in which only CDT traffic is processed. Note that 450 us is the minimum ST interval length (rounded up to 10us) that fulfills Eq. (4.30) on all switch ports in a network with synchronized switches. The second time interval corresponds to the ST cycle time, and is set to the period of ST streams, i.e. 5 ms. This implies that for ST intervals 450 us and 900 us in every 5 ms interval 4.55 ms and 4.1 ms (respectively) remain to process non-ST traffic streams, i.e. GCT, CAM, and NRT.

In this particular evaluation we use integration mode 1 so that the bandwidth loss caused by the guard bands of IEEE 802.1Qbv is comparatively small, as the guard band for CDT traffic only constitutes less than 3.6 % of the corresponding ST interval length and the guard band for the remaining non-ST traffic (GCT, CAM, and NRT) only constitutes less than 3.1 % of the remaining periodic 4.55 ms and 4.1 ms allocated to non-ST traffic (for 100 MBit/s links). Note that this overhead is small because in this evaluation setup we only need a single guard band every 5 ms, since we periodically schedule a ST interval of fixed length. In integration mode 2, however, where IEEE 802.1Qbv allows to schedule frames of each traffic stream in individual slots (cf. Section 4.4.2), without support for frame preemption, this would quickly result in a large overhead as will will evaluate in Section 5.5.

5.3.1.2 Shaper Configuration of IEEE 802.1Qch

For IEEE 802.1Qch, we investigate three shaper configurations characterized by their PS interval length: *IEEE 802.1Qch 280 us*, *IEEE 802.1Qch 560 us*, and *IEEE 802.1Qch 5 ms*. Where 560 us is the PS interval length that fulfills Eq. (4.50) (rounded up to 10us). For comparison, we also investigate a PS interval sizes of 280 us and 5 ms to illustrate the impact of the (artificial) delay induced by the PS interval length.

5.3.2 Results

Figures 5.4, 5.5, and 5.6 show, as boxplots, the worst-case end-to-end latency guarantees for IEEE 802.1Q, IEEE 802.1Qbv, and IEEE 802.1Qch for PCP4 (CDT traffic), PCP3 (GCT traffic), and PCP2 (CAM traffic), respectively. Here, we only show and discuss the quad star topology. The general results of our discussion, however, also apply to the other topologies (double star, tree, and line), which show similar results (cf. Appendix B.1).

In the following, we will discuss IEEE 802.1Qbv and IEEE 802.1Qch in separate subsections.

5.3.2.1 IEEE 802.1Qbv

We will discuss the impact of IEEE 802.1Qbv traffic shaping on high-priority CDT traffic streams, mid-priority GCT, and CAM traffic streams separately.

5.3.2.1.1 CDT Traffic

There is a significant difference between the unsynchronized and the synchronized configuration. In the unsynchronized configuration, we assume, that, in the worst-case, the frames of a ST stream just miss their ST intervals at each switch on their path through the network. This leads to long blocking times, which, in turn, increase the jitter. Larger jitter values can cause larger transient loads (bursts) on subsequent switch ports, which, in turn, require a longer time to be processed. In the worst-case this load cannot be processed in a single ST interval anymore, which leads to even larger blocking times and, thus, jitter. This can be observed for the *IEEE 802.1Qbv 450 us / 5 ms* configuration in Figure 5.4. Even though the shortest path for CDT traffic streams in the discussed quad star topology spans only two switches (cf. Appendix A), the shortest worst-case latency guarantee nearly extends over three full ST intervals. In the *IEEE 802.1Qbv 900 us / 5 ms* configuration, this effect is mitigated by the larger ST interval. Consequently, the latency guarantees for CDT traffic are lower in this configuration.

The synchronized configuration, in contrast, does not suffer from long blocking times, as we assume that all frames of ST streams arrive just in time for their respective ST intervals and, hence, only experience same-priority blocking. This results in significantly lower worst-case latency guarantees. In our setup, the largest worst-case latency guarantee for IEEE 802.1Qbv is 537 us, while, for IEEE 802.1Q, it is 1030 us.

Note however, while *IEEE 802.1Qbv 450 us / 5 ms sync* performs even better than IEEE 802.1Q, bear in mind that, here, we focus solely on the end-to-end latency guarantees in the Ethernet network. A system-wide end-to-end analysis, however, might involve additional components, e.g. the software communication stack in the ECUs. Also, there might be non-periodic latency-critical communication, such as spontaneous (e.g. emergency breaking) or angular-synchronized (e.g. engine control) communication. Hence, even if the Ethernet communication is synchronized, data might be delayed (and experience jitter) in other components, such that it might, in fact, not be (perfectly) synchronized to its designated ST interval.

5.3.2.1.2 GCT and CAM Traffic

Unshaped GCT and CAM traffic in IEEE 802.1Qbv on PCP 3 and PCP 2 (respectively) performs similar to IEEE 802.1Q, but the latency guarantees of these traffic streams are a bit larger than in IEEE 802.1Q. The reason for this is twofold: (1) Lower-priority (non-ST) streams experiences blocking by the ST intervals of ST streams regardless of whether these ST intervals are fully utilized or not (cf. Eq. (4.75)).[1] This is why the worst-case latency guarantees for GCT and CAM traffic are higher in the *IEEE 802.1Qbv 900 us / 5 ms* than in the *IEEE 802.1Qbv 450 us / 5 ms* and *IEEE 802.1Qbv 450 us / 5 ms sync* configurations. (2) Guard bands reduce the available bandwidth. In our setup, however, the guard bands of non-ST traffic only account for about less than 3.1% of the

[1] ST intervals might even be designed with an additional safety margin to allow for occasional overload.

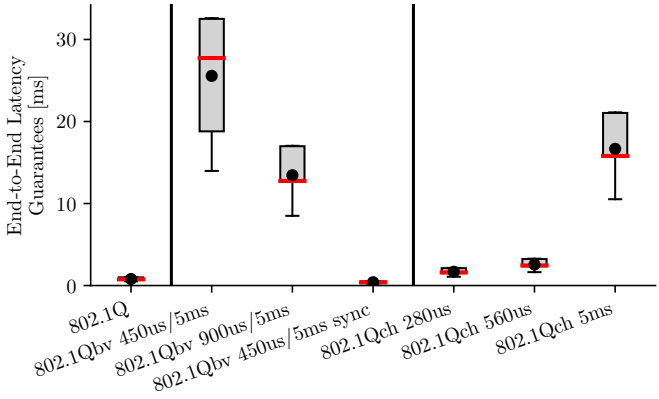


Figure 5.4: End-to-end latency guarantee comparison for PCP 4 traffic in the quad star topology

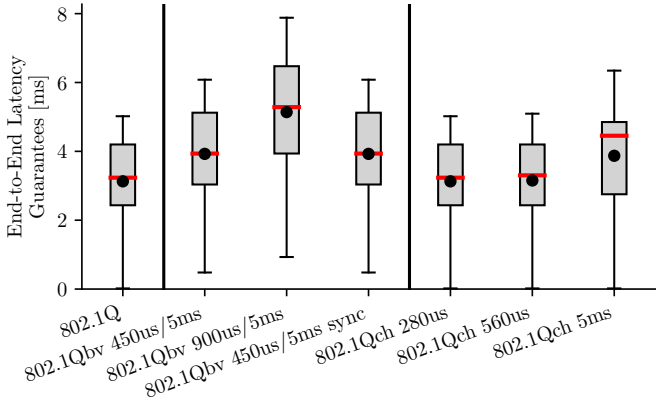


Figure 5.5: End-to-end latency guarantee comparison for PCP 3 traffic in the quad star topology

intervals in which non-ST streams are processed. Hence, the blocking by the ST intervals of ST traffic is the dominating factor. This is independent of whether network-wide gate schedule synchronization is assumed or not.

Note that, in the case of unsynchronized CDT traffic, the lower-priority GCT and CAM traffic has lower worst-case latency guarantees than the shaped high-priority CDT traffic.

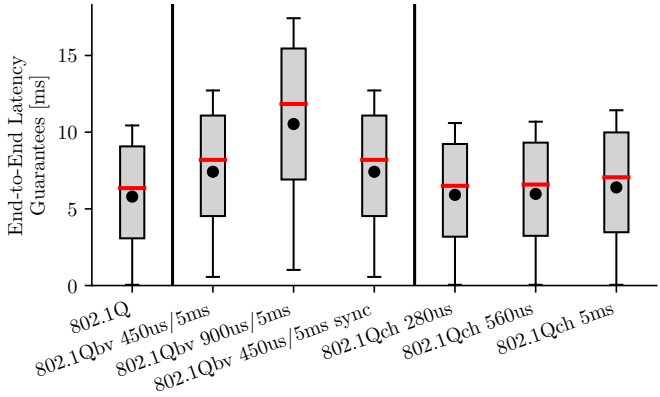


Figure 5.6: End-to-end latency guarantee comparison for PCP 2 traffic in the quad star topology

5.3.2.2 IEEE 802.1Qch

Again, we will discuss the impact of IEEE 802.1Qch traffic shaping on high-priority CDT traffic streams, mid-priority GCT, and CAM traffic streams separately.

5.3.2.2.1 CDT Traffic

For shaped high-priority traffic, we can clearly see the impact of the PS interval length on the worst-case latency guarantees. As IEEE 802.1Qch, in the worst-case, just artificially delays traffic by its PS interval length (and otherwise behaves like IEEE 802.1Q), shorter PS intervals lead to shorter delays and smaller jitter, which, in turn, result in lower worst-case latency guarantees. This is why, among our configurations, *IEEE 802.1Qch 280 us* performs best and *IEEE 802.1Qch 5 ms* performs worst.

Note that the artificial delay is clearly visible in Figure 5.4 for the *IEEE 802.1Qch 5 ms* configuration. As the shortest path for CDT traffic streams in the quad star topology spans only two switches (cf. Appendix A), the delay of the length of the PS interval is experienced at least twice (cf. IEEE 802.1Qch's design goal in Section 2.2.3.3). This is why the lowest worst-case latency guarantees start at around 10.5 ms.

For the *IEEE 802.1Qch 280 us* and *IEEE 802.1Qch 560 us* configurations, this effect is also visible, although normal frame processing and blocking have a higher relative share on the overall latency guarantees, i.e. for *IEEE 802.1Qch 280 us* and *IEEE 802.1Qch 560 us* the lowest worst-case latency guarantees start at around 1.1 ms and 1.6 ms, respectively.

5.3.2.2.2 GCT and CAM Traffic

Unshaped GCT and CAM traffic in IEEE 802.1Qch on PCP 3 and PCP 2 (respectively) performs similar to IEEE 802.1Q, but the latency guarantees of these traffic streams are a bit larger than in IEEE 802.1Q. The reason here is that higher-priority blocking from PS traffic streams is taken into account over integer multiples of ST intervals (cf. Eq. (4.32)). This is considerably visible in the *IEEE 802.1Qch 5 ms* configuration, which has the largest PS interval length.

Note that, in the case of the *IEEE 802.1Qch 5 ms* configuration, the lower-priority GCT and CAM traffic has lower worst-case latency guarantees than the shaped high-priority CDT traffic.

5.4 IEEE 802.1Q and IEEE 802.1Qbv with IEEE 802.3br and Integration Mode 1

In this and the following section we quantify the effect of frame preemption in Ethernet on the benefiting express traffic and on the discriminated preemptable traffic. In particular, we will investigate the effect of frame preemption on the worst-case latency guarantees for different sets of express and preemptable traffic classes in IEEE 802.1Q and IEEE 802.1Qbv. This section focuses on integration mode 1, while the next one focuses on integration mode 2.

5.4.1 Experimental Setup

The experimental setup follows the traffic partitioning into CDT, GCT, CAM, and NRT traffic and their priority mapping from Section 5.3.1. In order to quantify the effect of frame preemption, we consider different mappings of our CDT, GCT, CAM, and NRT traffic classes to express and preemptable traffic classes.

5.4.1.1 IEEE 802.1Q with Frame Preemption - Configurations

For IEEE 802.1Q we consider four different traffic class mappings, which are presented in Table 5.4. The configuration names are chosen to encode the traffic class mapping, i.e. *e43p21* indicates that PCPs 4 and 3 are mapped to the express traffic class and PCPs 2 and 1 are mapped to the preemptable traffic class. Note that configuration *e4321*, where all traffic classes are mapped to the express class, corresponds to plain IEEE 802.1Q.

5.4.1.2 IEEE 802.1Qbv with Frame Preemption - Configurations

For IEEE 802.1Qbv we consider two different traffic class mappings, which are presented in Table 5.5. In the configuration with frame preemption, only CDT traffic is mapped to the express MAC interface and GCT, CAM, and NRT are preemptable (as suggested by e.g. [180]). Mapping GCT to time-triggered slots would lead to very poor link utilization, due to the large variance of its

	Control Traffic		Camera Traffic	Background Traffic
	CDT	GCT	CAM	NRT
PCP	4	3	2	1
e4p321	express	preemptive	preemptive	preemptive
e43p21	express	express	preemptive	preemptive
e432p1	express	express	express	preemptive
e4321	express	express	express	express

Table 5.4: Priority code points and experiment description of IEEE 802.1Q with frame preemption

periods (cf. Appendix A). Again, configuration *e4321*, where all traffic classes are mapped to the express class, corresponds to plain IEEE 802.1Qbv.

In this evaluation, we only consider the *IEEE 802.1Qbv 450 us / 5 ms sync* configuration of the IEEE 802.1Qbv shaper (cf. Section 5.3.1.1).

	Control Traffic		Camera Traffic	Background Traffic
	CDT	GCT	CAM	NRT
PCP	4	3	2	1
e4p321	express	preemptive	preemptive	preemptive
e4321	express	express	express	express

Table 5.5: Priority code points and experiment description of IEEE 802.1Qbv with frame preemption

5.4.2 Results

Figures 5.7 and 5.8 show, as boxplots, the worst-case end-to-end latency guarantees for, IEEE 802.1Q and IEEE 802.1Qbv with frame preemption according to IEEE 802.3br, respectively. Again, we only show and discuss the quad star topology. The general results of our discussion, however, also apply to the other topologies as the double star, tree, and line topologies show similar results (cf. Appendix B.2).

As before, for all analyses, we use the optimization proposed in Section 4.7 (including the proposal for IEEE 802.1Qbv from Section 4.7.2).

In the following, we will discuss IEEE 802.1Qbv and IEEE 802.1Qch in separate subsections.

5.4.2.1 IEEE 802.1Q with Frame Preemption

The boxplots of Figure 5.7 are divided into three groups. The first group shows the results for CDT traffic under different mappings, while the second and third groups show the results for GCT and CAM traffic, respectively. By definition,

the end-to-end latencies of NRT traffic do not require timing verification and are omitted.

In general, the results match the expectations regarding frame preemption: (1) Mapping the streams of a traffic class to the express traffic class reduces their worst-case latency guarantees. Note however, that the amount of reduction also depends on the size of largest frame among the streams in the express traffic class, if traffic classes of multiple priorities are mapped to the express traffic class (cf. Eq. (4.51)). (2) Mapping the streams of a traffic class to the preemptable traffic class increase their worst-case latency guarantees. This increment depends on the number of interfering (higher-priority) express traffic streams. With more streams mapped to the express traffic class, the number of preemptions and the associated preemption overhead increases (cf. Eq. (4.63)). Thus the increment grows with an increasing number of express traffic streams.

Next we discuss each configuration from Table 5.4 in detail by comparing the peak latency guarantees. For *e4p321*, the worst-case end-to-end latency guarantees of CDT improve significantly over *e4321* (43.4 %). As expected, the worst-case latency guarantees of GCT and CAM are worse than under *e4321*, due to the additional preemption overhead. Their latency degradation, however, is comparatively small. On average GCT and CAM traffic have latency guarantees that are only 4.7 % and 2.7 % larger than under *e4321*, respectively.

In *e43p21*, the improvement of CDT is smaller than in *e4p321*, due to increased lower-priority blocking by GCT traffic, which cannot be preempted anymore. However, as GCT frames are comparatively small, CDT's average improvement is still 37.8 %. GCT, expectedly, improves and, compared to *e4321*, its worst-case latency guarantees are, on average, 8.9 % smaller. CAM traffic suffers from the additional preemption overhead from GCT and its (already large) latency guarantees are, on average, 14.7 % worse than in *e4321*.

In *e432p1*, the average improvements of CDT and GCT over *e4321* are 9.4 % and 2.2 % (respectively) as now they experience lower-priority blocking from large CAM frames, which are now also mapped to the express MAC. The average improvement of CAM traffic over *e4321* is 3.5 %.

Our experiments show that latency-critical (express) traffic clearly benefits from frame preemption. However, while the relative comparison suggests large improvements, the range of the absolute latency impact (reduction and increment) is very narrow. For CDT and GCT the peak worst-case latency guarantees are no further than 450 us apart and for CAM traffic they are no further than 1.54 ms apart.

5.4.2.2 IEEE 802.1Qbv with Frame Preemption

Figure 5.8 shows the results of these experiments. Again, the boxplots are divided into three groups showing the results for CDT, GCT, and CAM traffic.

As expected, time-triggered CDT traffic does not benefit from the preemption of GCT and CAM, as (plain) IEEE 802.1Qbv already ensures that time-triggered traffic is scheduled at predefined times. GCT and CAM traffic, however, benefits from frame preemption, even though they belong to the preemptable class. As GCT and CAM both use the preemptable MAC interface they can only be

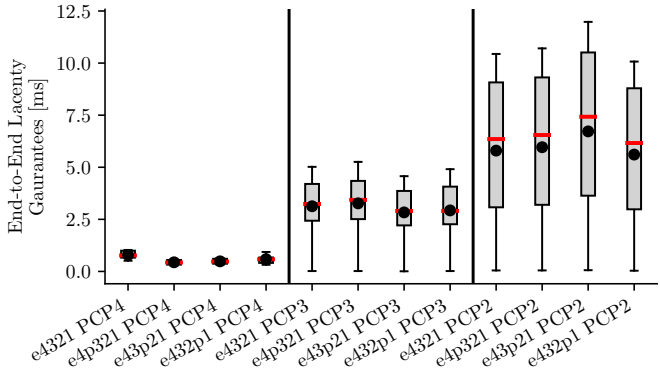


Figure 5.7: End-to-end latency guarantee comparison under IEEE 802.1Q/IEEE 802.3br frame preemption for different traffic mappings in the quad star topology and integration mode 1

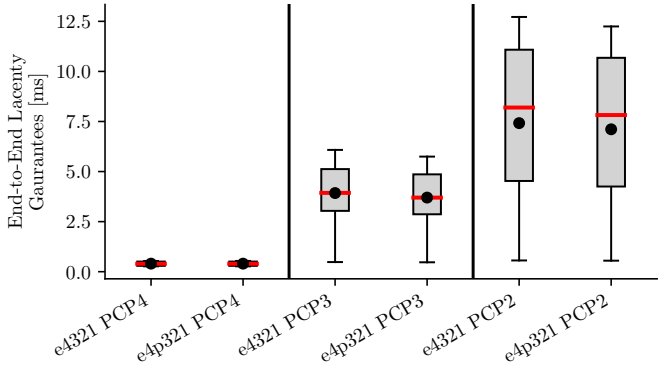


Figure 5.8: End-to-end latency guarantee comparison under IEEE 802.1Qbv/IEEE 802.3br frame preemption for different traffic mappings in the quad star topology and integration mode 1

preempted by express traffic, but not preempt themselves (Section 4.6). Frame preemption, however, reduces the interference from the time-triggered CDT slots by reducing the guard band (see Eq. (4.74)). In our setup, the guard band is reduced by 90 %. The preemption overhead is comparatively small, as there can only be one preemption per time-triggered slot, i.e. every 5 ms. More precisely, the worst-case end-to-end latency guarantees of GCT and CAM improve by 5.5 % and 3.7 % (peak values) compared to *e4321*, respectively.

As with IEEE 802.1Q and frame preemption, the absolute impact on the absolute latency guarantees is very narrow (the peak worst-case latency guarantees are no further than 510 us apart) and will be even less at gigabit speeds. However, in our setup, we assumed that all CDT traffic streams are mapped to one periodic ST interval. This is possible, as all CDT streams have the same period. Hence, the impact of the guard bands on the ST intervals and on the remaining time during which non-ST streams are transmitted is very small (less than 3 % (cf. Section 5.3.1)).

If, on the other hand, there are more ST intervals (e.g. for CDT traffic streams with different periods or even for individual frames) as in integration mode 2, then there are more guard bands to protect the ST intervals and these guard bands can have a more significant impact on the time during which non-ST streams are scheduled. We will investigate this in Section 5.5.

5.4.2.3 Comparison of Worst-Case End-to-End Latency Guarantees

Comparing the results for IEEE 802.1Q and IEEE 802.1Qbv yields an interesting finding: As long as CAM traffic is not using the express MAC interface (i.e. stays preemptable), the worst-case end-to-end latency guarantees of CDT traffic are almost identical for IEEE 802.1Q and IEEE 802.1Qbv (*e4p321* and *e43p21* in Figure 5.7 vs. *e4p321* in Figure 5.8). For *e4p321*, their maximum difference is 46 us and for *e43p21* it is 104 us (i.e. between 0.92 % and 2 % of the CDT period), which is very low compared to typical automotive latency requirements. This is because frame preemption in IEEE 802.1Q reduces the lower-priority blocking for express traffic to the technical minimum (Eq. (4.51)). This improvement vanishes, if long frames, e.g. from CAM traffic in *e432p1*, are introduced to the express traffic class (which can, again, be explained by Eq. (4.51)). Comparing the worst-case latency guarantees of GCT and CAM traffic between IEEE 802.1Q (PCP3 and PCP2 in Figure 5.7) and IEEE 802.1Qbv (PCP3 and PCP2 in Figure 5.8), we see that, under IEEE 802.1Qbv, these latencies are worse than under IEEE 802.1Q.

Without frame preemption, similar lower-priority blocking times could be achieved by permanently limiting the length of lower-priority frames. This, however, would entail a large overhead as more Ethernet frames are required and each additional frame requires 42 bytes of protocol overhead (each frame must include preamble, SFD, DA, SA, Q-Tag, ET, FCS, and IFG, see Figure 4.6). Frame preemption, in contrast, only splits frames into shorter fragments when required. As the periods of control traffic (CDT and GCT) are typically larger than those of CAM traffic (see Appendix A), not all CAM frames will be preempted.

5.5 IEEE 802.1Q and IEEE 802.1Qbv with IEEE 802.3br and Integration Mode 2

In this section we quantify the effect of frame preemption in Ethernet on the worst-case end-to-end latency guarantees for different sets of express

and preemptable traffic classes in IEEE 802.1Q and IEEE 802.1Qbv when integration mode 2 is used.

While integration mode 1 showed only marginal improvements of the worst-case end-to-end latency guarantees of non-CDT traffic under IEEE 802.1Qbv with frame preemption compared to the scenario without frame preemption (cf. Figure 5.8), we expect to observe much higher improvement from integration mode 2, due to the increased number of required guard bands.

5.5.1 Experimental Setup

The experimental setup is based on Section 5.4.1. In the first attempt, we tried to use exactly the same setup as in Section 5.4.1 but use integration mode 2 for all CDT traffic streams, i.e. configuring each CDT frame to use its own dedicated ST slot. With this setup, however, the non-preemptive IEEE 802.1Qbv scenarios became unscheduable, due to poor link utilization caused by the excessive number of guard bands. Up to 20 different CDT frames (each with their own ST slot) may be transferred over a single 100 MBit/s Ethernet link in the quad star topology. With an individual guard band overhead of 3.1 % per ST slot, these 20 ST slots cause an accumulated guard band overhead of up to 62 %, during which no frames are allowed to be transmitted. With frame preemption, the individual guard band overhead decreases to 0.5 %, resulting in a significantly lower accumulated guard band overhead of 10 %.

In order to create a scenario, where we are able to compare a network and its traffic with and without frame preemption, we reduced the number of CDT traffic streams by keeping the traffic description from Table A.1 but only considering unicast traffic streams with a period of 5 ms and a payload of 8 bytes to be CDT traffic streams.

5.5.2 Results

The focus of this section is the impact of integration mode 2 on IEEE 802.1Q and IEEE 802.1Qbv with frame preemption. Hence, we will not provide a detailed comparison as in Section 5.4.2, but instead focus on impact of frame preemption on integration mode 2 exclusively.

5.5.2.1 IEEE 802.1Q with Frame Preemption

Comparing Figures 5.7 and 5.9, we see that the end-to-end latency guarantees are a bit lower in integration mode 2, but the general behavior of IEEE 802.1Q with frame preemption in integration mode 2 is very similar to that of integration mode 1. This is what we expected, because in this setup we have less CDT traffic and because there are no guard bands in IEEE 802.1Q with frame preemption.

5.5.2.2 IEEE 802.1Qbv with Frame Preemption

Comparing *e4321 PCP3* and *e4321 PCP2* to *e4p321 PCP3* and *e4p321 PCP2* in Figure 5.10 (respectively), we see that non-CDT traffic benefits significantly

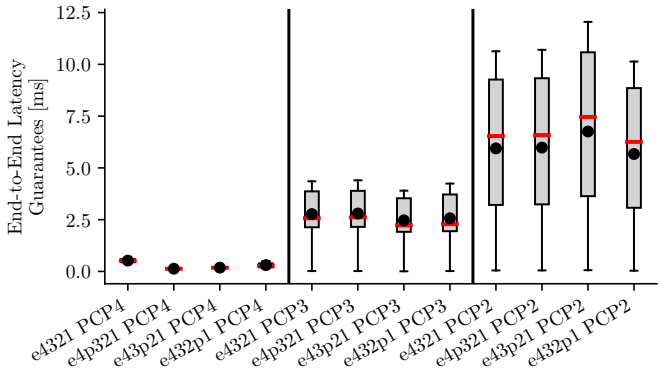


Figure 5.9: End-to-end latency guarantee comparison under IEEE 802.1Q/IEEE 802.3br frame preemption for different traffic mappings in the quad star topology and integration mode 2

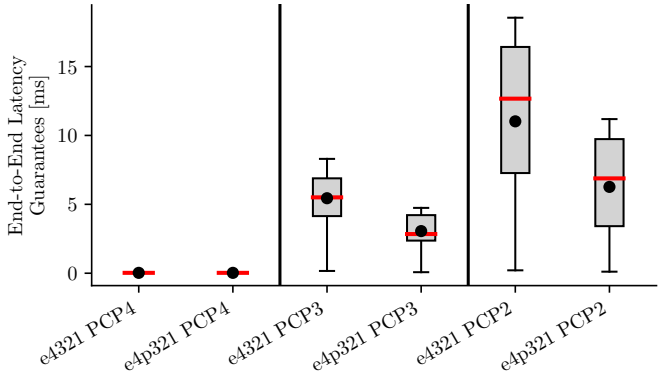


Figure 5.10: End-to-end latency guarantee comparison under IEEE 802.1Qbv/IEEE 802.3br frame preemption for different traffic mappings in the quad star topology and integration mode 2

from frame preemption in integration mode 2 (even though we already reduced the number of CDT traffic streams and increased the number of GCD traffic streams). Table 5.6 summarizes the reduction of the worst-case end-to-end latency guarantees for non-CDT traffic on PCP 3 and PCP 2 in percent.

Futhermore, we see, when comparing the worst-case latency guarantees for GCD and CAM traffic between IEEE 802.1Q and IEEE 802.1Qbv in Figures 5.9 and 5.10, that the worst-case latency guarantees for both GCD and CAM traffic

	Double Star	Quad Star	Tree	Line
PCP 3	30.50 %	42.86 %	41.81 %	51.77 %
PCP 2	31.92 %	39.68 %	37.19 %	54.38 %

Table 5.6: Peak improvements with frame preemption under IEEE 802.1Qbv in integration mode 2

are always worse under IEEE 802.1Qbv, even with frame preemption. Although frame preemption significantly closes the gap.

5.6 Summary

In this chapter, we focused on the evaluation of our proposed formal timing analysis methods. To this end, we defined a set of traffic streams of different priority and characteristics (control, camera, non-real-time) and used a set of typical automotive Ethernet topologies. The evaluation metric were the worst-case end-to-end latency guarantees provided by our formal timing analysis.

First, we evaluated our proposed formal timing analysis from Chapter 4 and the proposed optimization, i.e. exploiting workload correlation among traffic streams, from Section 4.7 by using different network topologies. We saw that there is a clear impact of both the priority of a traffic stream as well as the network topology on its worst-case latency guarantees. Furthermore, we could empirically verify the usefulness of our proposed optimization by showing that it yields significantly lower worst-case latency guarantees.

Next, we evaluated the worst-case latency guarantees of Ethernet TSN's IEEE 802.1Qbv and IEEE 802.1Qch shapers for latency-critical traffic, as well as their impact on unshaped lower-priority traffic. Our experiments show that, IEEE 802.1Qbv can give very low latency guarantees when all shapers are synchronized. However, it suffers from long blocking times if this cannot be guaranteed, which underlines the need for reliable time synchronization in IEEE 802.1Qbv setups.[2] IEEE 802.1Qch suffers from the (artificial) delay caused by its PS interval. We showed, that the worst-case performance of IEEE 802.1Qch is always worse than that of IEEE 802.1Q.

In the context of frame preemption according to IEEE 802.3br, we evaluated IEEE 802.1Q and IEEE 802.1Qbv. For IEEE 802.1Q, we showed that, higher priority traffic benefits from frame preemption. The absolute benefit, however, was fairly narrow. In IEEE 802.1Qbv, non-ST traffic streams benefit from frame preemption due to the reduced guard band length. However, in integration mode 1, the improvement was very small, as there is only one guard band per ST interval. It is integration mode 2 that exposes the real benefit of frame preemption, due to the larger number of guard bands per ST interval that are reduced. In our setup, we saw improvements between 30 % and 50 %, for preempted traffic depending on the topology.

[2]Including protection against attacks such as tempering.

Chapter 6

Conclusion

Ethernet has been chosen to become the backbone network of current and future automotive E/E architectures, due to its scalability, high speed, and low cost. Modern in-vehicle networks must accommodate many different communication use cases, in particular traffic from highly demanding ADAS applications, which are a key building block of (future) autonomously driving vehicles, as well as Car-to-X and customer Internet traffic. This results in diverse mixed-critical communication requirements and calls for appropriate quality of service mechanisms on Ethernet level. These mechanisms (among other things) are addressed by the Ethernet AVB and Ethernet TSN set of standards.

In this thesis we presented formal methods to evaluate the performance of Ethernet TSN standards under worst-case conditions based on the compositional performance analysis framework [66]. Our focus was on the transmission selection mechanisms of IEEE 802.1Qbv (enhancements for scheduled traffic), IEEE 802.1Qch (cyclic queueing and forwarding), and IEEE 802.3br/IEEE 802.1Qbu (frame preemption). For our evaluation, we considered four typical automotive network topologies (double star, quad star, tree, and line) and derived formal performance guarantees for various traffic classes (control, video, best effort) under different quality of service mechanisms and configurations. The worst-case guarantees of the end-to-end latencies were used as the comparison metric.

First, we showed the significant end-to-end latency guarantee improvement (i.e. reduction) of exploiting workload correlations for IEEE 802.1Q. We also discussed the impact of topology and traffic priority on the correlation potential. As the correlations exploit common predecessor resources along a path, larger topologies bear more potential for improvement. From a priority (or traffic class) perspective, there are two key factors: the number of traffic streams and the priority. The large number of control traffic streams in our setup offers more correlation potential than the smaller number of video traffic streams. As higher-priority traffic is always transmitted before lower-priority traffic, the potential for improvement decreases with decreasing priority. This optimization has been implemented for all our Ethernet performance analyses.

Next, we compared the formal end-to-end latency guarantees from our proposed analysis of IEEE 802.1Q, IEEE 802.1Qbv, and IEEE 802.1Qch under different transmission selection configurations. For IEEE 802.1Qbv, we investigated a time-synchronized configuration where the scheduled traffic interval is of the minimum required size and two unsynchronized configurations, one where the scheduled traffic interval is of minimum required size and one where it is twice as long. We saw that reliable time synchronization throughout the entire communication path is a must have (including the producer and consumer software stacks) in order to avoid large sampling delays for scheduled traffic. Compared to IEEE 802.1Q, non-scheduled traffic experiences more interference from scheduled traffic, as scheduled traffic intervals might not (always) be fully utilized and because of the additional guard bands, which, especially with multiple scheduled traffic intervals, can greatly decrease performance. For IEEE 802.1Qch, we investigated three different peristaltic interval sizes. We saw that, without synchronization of the peristaltic intervals, the additional delay to wait for the next peristaltic interval is clearly visible in the end-to-end latency guarantees of shaped traffic, while non-shaped traffic performs similar to IEEE 802.1Q.

Finally, we evaluated the impact of frame preemption for IEEE 802.1Q and IEEE 802.1Qbv under different preemption configurations (i.e. which traffic streams are preemptable). For IEEE 802.1Q the absolute benefit of frame preemption was measurable but very narrow for the evaluated scenarios, as only the (single-instance) lower-priority blocking is reduced. For IEEE 802.1Qbv, non-scheduled traffic streams benefit from frame preemption, as preemption reduces the length of the guard bands. Especially in scenarios where there are multiple scheduled intervals and, hence, multiple guard bands, there is significantly more bandwidth left for non-scheduled traffic streams.

Overall, our results apply to all evaluated topologies, with the larger ones showing the effects of workload correlation and different quality of service mechanisms more clearly.

6.1 Directions for Future Research

Ethernet and Ethernet TSN, in particular, are evolving standards in the sense that amendments introducing new technologies and features are added to the main standards. In addition to the formal analyses presented in this thesis, there are new transmission selection algorithms to consider, such as IEEE 802.1Qcr [82], or new physical layers, such as the shared-medium-based IEEE 802.3cg [5] for 10 MBit/s. Quantifying the worst-case timing impact of IEEE 802.1CB [78] would also be a great benefit for fail-operational use cases (while taking the challenges outlined in [143] into account).

Another source of communication latency is IP routing. Automotive Ethernet networks are often segmented into different subnets, e.g. to group similar functions or to implement security domains. An IP router (typically including a firewall) is used to route traffic streams between these subnets. IP routers can be implemented in software (e.g. as part of the firmware running on the management microcontroller of a switch) or in hardware. In the former case, the

microcontroller must be considered as a (potentially shared) resource during performance analysis.

Consolidation and performance requirements, especially in the centralized and zonal E/E architectures, introduce new technologies, that need to be covered during network design of time-critical systems. In particular, the network extends into the (SoC) chips, yielding new kinds of switches, such as software-based virtual switches competing for CPU time with other functions [185] and hardware-based switches utilizing PCIe SR-IOV, where multiple lightweight virtual network interfaces are multiplexed through a shared PCIe link.

SDN [107] and SDN-like functions, e.g. [87], [86], could become interesting in the context of dynamic network (re-)configuration, e.g. to mitigate failures or implement resource broking [131], [28]. Here, the overhead of the control plane must be analyzed. First work towards this direction has been conducted in [168].

Appendix A

Detailed Traffic Description

A.1 Traffic Description

In Table A.1, we present a detailed description of the traffic used during evaluation. The table includes a symbolic traffic stream name, the source of the traffic stream, and its destination. Multicast and broadcast traffic streams have multiple destinations. A traffic stream's priority is given by the PCP field used in the IEEE 802.1Q-tag. Together the protocol and payload columns specify the resulting frame size: payload specifies the raw (application) payload, while the protocol specifies the protocol overhead in addition to the raw payload. Period, jitter, and dmin define the traffic stream's event model in the form of a parameterized PJd event model (cf. Section 3.2.2.2). The initial minimum distance (dmin) between two frames is set to the frame length[1].

Traffic Stream	Source	Destination(s)	PCP	Protocol	Payload	Period	Jitter	Dmin
MessageECU0#0	ECU0	ECU6	3	IPv4+UDP	8.0	5 ms	5 ms	0.672 us
MessageECU0#1	ECU0	ECU4	3	IPv4+UDP	8.0	10 ms	10 ms	0.672 us
MessageECU0#2	ECU0	ECU6	3	IPv4+UDP	8.0	20 ms	20 ms	0.672 us
MessageECU0#3	ECU0	ECU4	3	IPv4+UDP	8.0	50 ms	50 ms	0.672 us
MessageECU0#4	ECU0	ECU6	3	IPv4+UDP	8.0	100 ms	100 ms	0.672 us
MessageECU0#5	ECU0	ECU4	3	IPv4+UDP	8.0	200 ms	200 ms	0.672 us
MessageECU0#6	ECU0	ECU6	3	IPv4+UDP	64.0	5 ms	5 ms	1.072 us
MessageECU0#7	ECU0	ECU4	3	IPv4+UDP	64.0	10 ms	10 ms	1.072 us
MessageECU0#8	ECU0	ECU6	3	IPv4+UDP	64.0	20 ms	20 ms	1.072 us
MessageECU0#9	ECU0	ECU4	3	IPv4+UDP	64.0	50 ms	50 ms	1.072 us
MessageECU0#10	ECU0	ECU6	3	IPv4+UDP	64.0	100 ms	100 ms	1.072 us
MessageECU0#11	ECU0	ECU4	3	IPv4+UDP	64.0	200 ms	200 ms	1.072 us
MessageECU0#12	ECU0	ECU6	3	IPv4+UDP	96.0	5 ms	5 ms	1.328 us
MessageECU0#13	ECU0	ECU4	3	IPv4+UDP	96.0	10 ms	10 ms	1.328 us
MessageECU0#14	ECU0	ECU6	3	IPv4+UDP	96.0	20 ms	20 ms	1.328 us
MessageECU0#15	ECU0	ECU4	3	IPv4+UDP	96.0	50 ms	50 ms	1.328 us
MessageECU0#16	ECU0	ECU6	3	IPv4+UDP	96.0	100 ms	100 ms	1.328 us
MessageECU0#17	ECU0	ECU4	3	IPv4+UDP	128.0	5 ms	5 ms	1.584 us
MessageECU0#18	ECU0	ECU6	3	IPv4+UDP	128.0	10 ms	10 ms	1.584 us
MessageECU0#19	ECU0	ECU4	3	IPv4+UDP	160.0	10 ms	10 ms	1.84 us
MessageECU0#20	ECU0	ECU6	3	IPv4+UDP	192.0	50 ms	50 ms	2.096 us
MessageECU0#21	ECU0	ECU4	3	IPv4+UDP	192.0	100 ms	100 ms	2.096 us

[1] Even with frame preemption, the minimum distance between any two frames of the same traffic stream cannot be smaller.

Traffic Stream	Source	Destination(s)	PCP	Protocol	Payload	Period	Jitter	Dmin
MessageECU0#22	ECU0	ECU6	3	IPv4+UDP	256.0	50 ms	50 ms	2.608 us
MessageECU0#23	ECU0	ECU4	3	IPv4+UDP	256.0	100 ms	100 ms	2.608 us
MessageECU0#24	ECU0	ECU4, ECU6	3	IPv4+UDP	8.0	5 ms	5 ms	0.672 us
MessageECU0#25	ECU0	ECU2, ECU4, ECU6	3	IPv4+UDP	64.0	200 ms	200 ms	1.072 us
MessageECU0#26	ECU0	ECU2, ECU4, ECU6, ECU7	3	IPv4+UDP	256.0	50 ms	50 ms	2.608 us
MessageECU0#27	ECU0	BROADCAST	3	IPv4+UDP	32.0	20 ms	20 ms	0.816 us
MessageECU0#28	ECU0	BROADCAST	3	IPv4+UDP	64.0	50 ms	50 ms	1.072 us
MessageECU0#29	ECU0	BROADCAST	3	IPv4+UDP	128.0	100 ms	100 ms	1.584 us
MessageECU1#0	ECU1	ECU6	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU1#1	ECU1	ECU4	3	IPv4+UDP	8.0	10 ms	10 ms	6.72 us
MessageECU1#2	ECU1	ECU6	3	IPv4+UDP	8.0	20 ms	20 ms	6.72 us
MessageECU1#3	ECU1	ECU4	3	IPv4+UDP	8.0	50 ms	50 ms	6.72 us
MessageECU1#4	ECU1	ECU6	3	IPv4+UDP	8.0	100 ms	100 ms	6.72 us
MessageECU1#5	ECU1	ECU4	3	IPv4+UDP	8.0	200 ms	200 ms	6.72 us
MessageECU1#6	ECU1	ECU6	3	IPv4+UDP	64.0	5 ms	5 ms	10.72 us
MessageECU1#7	ECU1	ECU4	3	IPv4+UDP	64.0	10 ms	10 ms	10.72 us
MessageECU1#8	ECU1	ECU6	3	IPv4+UDP	64.0	20 ms	20 ms	10.72 us
MessageECU1#9	ECU1	ECU4	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU1#10	ECU1	ECU6	3	IPv4+UDP	64.0	100 ms	100 ms	10.72 us
MessageECU1#11	ECU1	ECU4	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU1#12	ECU1	ECU6	3	IPv4+UDP	96.0	5 ms	5 ms	13.28 us
MessageECU1#13	ECU1	ECU4	3	IPv4+UDP	96.0	10 ms	10 ms	13.28 us
MessageECU1#14	ECU1	ECU6	3	IPv4+UDP	96.0	20 ms	20 ms	13.28 us
MessageECU1#15	ECU1	ECU4	3	IPv4+UDP	96.0	50 ms	50 ms	13.28 us
MessageECU1#16	ECU1	ECU6	3	IPv4+UDP	96.0	100 ms	100 ms	13.28 us
MessageECU1#17	ECU1	ECU4	3	IPv4+UDP	128.0	5 ms	5 ms	15.84 us
MessageECU1#18	ECU1	ECU6	3	IPv4+UDP	128.0	10 ms	10 ms	15.84 us
MessageECU1#19	ECU1	ECU4	3	IPv4+UDP	160.0	10 ms	10 ms	18.4 us
MessageECU1#20	ECU1	ECU6	3	IPv4+UDP	192.0	50 ms	50 ms	20.96 us
MessageECU1#21	ECU1	ECU4	3	IPv4+UDP	192.0	100 ms	100 ms	20.96 us
MessageECU1#22	ECU1	ECU6	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU1#23	ECU1	ECU4	3	IPv4+UDP	256.0	100 ms	100 ms	26.08 us
MessageECU1#24	ECU1	ECU4, ECU6	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU1#25	ECU1	ECU2, ECU4, ECU6	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU1#26	ECU1	ECU2, ECU4, ECU6, ECU7	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU1#27	ECU1	BROADCAST	3	IPv4+UDP	32.0	20 ms	20 ms	8.16 us
MessageECU1#28	ECU1	BROADCAST	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU1#29	ECU1	BROADCAST	3	IPv4+UDP	128.0	100 ms	100 ms	15.84 us
MessageECU2#0	ECU2	ECU6	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU2#1	ECU2	ECU4	3	IPv4+UDP	8.0	10 ms	10 ms	6.72 us
MessageECU2#2	ECU2	ECU6	3	IPv4+UDP	8.0	20 ms	20 ms	6.72 us
MessageECU2#3	ECU2	ECU4	3	IPv4+UDP	8.0	50 ms	50 ms	6.72 us
MessageECU2#4	ECU2	ECU6	3	IPv4+UDP	8.0	100 ms	100 ms	6.72 us
MessageECU2#5	ECU2	ECU4	3	IPv4+UDP	8.0	200 ms	200 ms	6.72 us
MessageECU2#6	ECU2	ECU6	3	IPv4+UDP	64.0	5 ms	5 ms	10.72 us
MessageECU2#7	ECU2	ECU4	3	IPv4+UDP	64.0	10 ms	10 ms	10.72 us
MessageECU2#8	ECU2	ECU6	3	IPv4+UDP	64.0	20 ms	20 ms	10.72 us
MessageECU2#9	ECU2	ECU4	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU2#10	ECU2	ECU6	3	IPv4+UDP	64.0	100 ms	100 ms	10.72 us
MessageECU2#11	ECU2	ECU4	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU2#12	ECU2	ECU6	3	IPv4+UDP	96.0	5 ms	5 ms	13.28 us
MessageECU2#13	ECU2	ECU4	3	IPv4+UDP	96.0	10 ms	10 ms	13.28 us
MessageECU2#14	ECU2	ECU6	3	IPv4+UDP	96.0	20 ms	20 ms	13.28 us
MessageECU2#15	ECU2	ECU4	3	IPv4+UDP	96.0	50 ms	50 ms	13.28 us
MessageECU2#16	ECU2	ECU6	3	IPv4+UDP	96.0	100 ms	100 ms	13.28 us
MessageECU2#17	ECU2	ECU4	3	IPv4+UDP	128.0	5 ms	5 ms	15.84 us
MessageECU2#18	ECU2	ECU6	3	IPv4+UDP	128.0	10 ms	10 ms	15.84 us
MessageECU2#19	ECU2	ECU4	3	IPv4+UDP	160.0	10 ms	10 ms	18.4 us
MessageECU2#20	ECU2	ECU6	3	IPv4+UDP	192.0	50 ms	50 ms	20.96 us
MessageECU2#21	ECU2	ECU4	3	IPv4+UDP	192.0	100 ms	100 ms	20.96 us
MessageECU2#22	ECU2	ECU6	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU2#23	ECU2	ECU4	3	IPv4+UDP	256.0	100 ms	100 ms	26.08 us
MessageECU2#24	ECU2	ECU4, ECU6	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU2#25	ECU2	ECU1, ECU4, ECU6	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU2#26	ECU2	ECU1, ECU4, ECU6, ECU7	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU2#27	ECU2	BROADCAST	3	IPv4+UDP	32.0	20 ms	20 ms	8.16 us
MessageECU2#28	ECU2	BROADCAST	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU2#29	ECU2	BROADCAST	3	IPv4+UDP	128.0	100 ms	100 ms	15.84 us
MessageECU3#0	ECU3	ECU6	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU3#1	ECU3	ECU4	3	IPv4+UDP	8.0	10 ms	10 ms	6.72 us
MessageECU3#2	ECU3	ECU6	3	IPv4+UDP	8.0	20 ms	20 ms	6.72 us

Traffic Stream	Source	Destination(s)	PCP	Protocol	Payload	Period	Jitter	Dmin
MessageECU3#3	ECU3	ECU4	3	IPv4+UDP	8.0	50 ms	50 ms	6.72 us
MessageECU3#4	ECU3	ECU6	3	IPv4+UDP	8.0	100 ms	100 ms	6.72 us
MessageECU3#5	ECU3	ECU4	3	IPv4+UDP	8.0	200 ms	200 ms	6.72 us
MessageECU3#6	ECU3	ECU6	3	IPv4+UDP	64.0	5 ms	5 ms	10.72 us
MessageECU3#7	ECU3	ECU4	3	IPv4+UDP	64.0	10 ms	10 ms	10.72 us
MessageECU3#8	ECU3	ECU6	3	IPv4+UDP	64.0	20 ms	20 ms	10.72 us
MessageECU3#9	ECU3	ECU4	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU3#10	ECU3	ECU6	3	IPv4+UDP	64.0	100 ms	100 ms	10.72 us
MessageECU3#11	ECU3	ECU4	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU3#12	ECU3	ECU6	3	IPv4+UDP	96.0	5 ms	5 ms	13.28 us
MessageECU3#13	ECU3	ECU4	3	IPv4+UDP	96.0	10 ms	10 ms	13.28 us
MessageECU3#14	ECU3	ECU6	3	IPv4+UDP	96.0	20 ms	20 ms	13.28 us
MessageECU3#15	ECU3	ECU4	3	IPv4+UDP	96.0	50 ms	50 ms	13.28 us
MessageECU3#16	ECU3	ECU6	3	IPv4+UDP	96.0	100 ms	100 ms	13.28 us
MessageECU3#17	ECU3	ECU4	3	IPv4+UDP	128.0	5 ms	5 ms	15.84 us
MessageECU3#18	ECU3	ECU6	3	IPv4+UDP	128.0	10 ms	10 ms	15.84 us
MessageECU3#19	ECU3	ECU4	3	IPv4+UDP	160.0	10 ms	10 ms	18.4 us
MessageECU3#20	ECU3	ECU6	3	IPv4+UDP	192.0	50 ms	50 ms	20.96 us
MessageECU3#21	ECU3	ECU4	3	IPv4+UDP	192.0	100 ms	100 ms	20.96 us
MessageECU3#22	ECU3	ECU6	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU3#23	ECU3	ECU4	3	IPv4+UDP	256.0	100 ms	100 ms	26.08 us
MessageECU3#24	ECU3	ECU4, ECU6	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU3#25	ECU3	ECU1, ECU4, ECU6	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU3#26	ECU3	ECU1, ECU4, ECU6, ECU7	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU3#27	ECU3	BROADCAST	3	IPv4+UDP	32.0	20 ms	20 ms	8.16 us
MessageECU3#28	ECU3	BROADCAST	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU3#29	ECU3	BROADCAST	3	IPv4+UDP	128.0	100 ms	100 ms	15.84 us
MessageECU4#0	ECU4	ECU1	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU4#1	ECU4	ECU3	3	IPv4+UDP	8.0	10 ms	10 ms	6.72 us
MessageECU4#2	ECU4	ECU1	3	IPv4+UDP	8.0	20 ms	20 ms	6.72 us
MessageECU4#3	ECU4	ECU3	3	IPv4+UDP	8.0	50 ms	50 ms	6.72 us
MessageECU4#4	ECU4	ECU1	3	IPv4+UDP	8.0	100 ms	100 ms	6.72 us
MessageECU4#5	ECU4	ECU3	3	IPv4+UDP	8.0	200 ms	200 ms	6.72 us
MessageECU4#6	ECU4	ECU1	3	IPv4+UDP	64.0	5 ms	5 ms	10.72 us
MessageECU4#7	ECU4	ECU3	3	IPv4+UDP	64.0	10 ms	10 ms	10.72 us
MessageECU4#8	ECU4	ECU1	3	IPv4+UDP	64.0	20 ms	20 ms	10.72 us
MessageECU4#9	ECU4	ECU3	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU4#10	ECU4	ECU1	3	IPv4+UDP	64.0	100 ms	100 ms	10.72 us
MessageECU4#11	ECU4	ECU3	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU4#12	ECU4	ECU1	3	IPv4+UDP	96.0	5 ms	5 ms	13.28 us
MessageECU4#13	ECU4	ECU3	3	IPv4+UDP	96.0	10 ms	10 ms	13.28 us
MessageECU4#14	ECU4	ECU1	3	IPv4+UDP	96.0	20 ms	20 ms	13.28 us
MessageECU4#15	ECU4	ECU3	3	IPv4+UDP	96.0	50 ms	50 ms	13.28 us
MessageECU4#16	ECU4	ECU1	3	IPv4+UDP	96.0	100 ms	100 ms	13.28 us
MessageECU4#17	ECU4	ECU3	3	IPv4+UDP	128.0	5 ms	5 ms	15.84 us
MessageECU4#18	ECU4	ECU1	3	IPv4+UDP	128.0	10 ms	10 ms	15.84 us
MessageECU4#19	ECU4	ECU3	3	IPv4+UDP	160.0	10 ms	10 ms	18.4 us
MessageECU4#20	ECU4	ECU1	3	IPv4+UDP	192.0	50 ms	50 ms	20.96 us
MessageECU4#21	ECU4	ECU3	3	IPv4+UDP	192.0	100 ms	100 ms	20.96 us
MessageECU4#22	ECU4	ECU1	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU4#23	ECU4	ECU3	3	IPv4+UDP	256.0	100 ms	100 ms	26.08 us
MessageECU4#24	ECU4	ECU3, ECU1	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU4#25	ECU4	ECU6, ECU3, ECU1	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU4#26	ECU4	ECU6, ECU3, ECU1, ECU0	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU4#27	ECU4	BROADCAST	3	IPv4+UDP	32.0	20 ms	20 ms	8.16 us
MessageECU4#28	ECU4	BROADCAST	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU4#29	ECU4	BROADCAST	3	IPv4+UDP	128.0	100 ms	100 ms	15.84 us
MessageECU5#0	ECU5	ECU1	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU5#1	ECU5	ECU3	3	IPv4+UDP	8.0	10 ms	10 ms	6.72 us
MessageECU5#2	ECU5	ECU1	3	IPv4+UDP	8.0	20 ms	20 ms	6.72 us
MessageECU5#3	ECU5	ECU3	3	IPv4+UDP	8.0	50 ms	50 ms	6.72 us
MessageECU5#4	ECU5	ECU1	3	IPv4+UDP	8.0	100 ms	100 ms	6.72 us
MessageECU5#5	ECU5	ECU3	3	IPv4+UDP	8.0	200 ms	200 ms	6.72 us
MessageECU5#6	ECU5	ECU1	3	IPv4+UDP	64.0	5 ms	5 ms	10.72 us
MessageECU5#7	ECU5	ECU3	3	IPv4+UDP	64.0	10 ms	10 ms	10.72 us
MessageECU5#8	ECU5	ECU1	3	IPv4+UDP	64.0	20 ms	20 ms	10.72 us
MessageECU5#9	ECU5	ECU3	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU5#10	ECU5	ECU1	3	IPv4+UDP	64.0	100 ms	100 ms	10.72 us
MessageECU5#11	ECU5	ECU3	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU5#12	ECU5	ECU1	3	IPv4+UDP	96.0	5 ms	5 ms	13.28 us
MessageECU5#13	ECU5	ECU3	3	IPv4+UDP	96.0	10 ms	10 ms	13.28 us

Traffic Stream	Source	Destination(s)	PCP	Protocol	Payload	Period	Jitter	Dmin
MessageECU5#14	ECU5	ECU1	3	IPv4+UDP	96.0	20 ms	20 ms	13.28 us
MessageECU5#15	ECU5	ECU3	3	IPv4+UDP	96.0	50 ms	50 ms	13.28 us
MessageECU5#16	ECU5	ECU1	3	IPv4+UDP	96.0	100 ms	100 ms	13.28 us
MessageECU5#17	ECU5	ECU3	3	IPv4+UDP	128.0	5 ms	5 ms	15.84 us
MessageECU5#18	ECU5	ECU1	3	IPv4+UDP	128.0	10 ms	10 ms	15.84 us
MessageECU5#19	ECU5	ECU3	3	IPv4+UDP	160.0	10 ms	10 ms	18.4 us
MessageECU5#20	ECU5	ECU1	3	IPv4+UDP	192.0	50 ms	50 ms	20.96 us
MessageECU5#21	ECU5	ECU3	3	IPv4+UDP	192.0	100 ms	100 ms	20.96 us
MessageECU5#22	ECU5	ECU1	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU5#23	ECU5	ECU3	3	IPv4+UDP	256.0	100 ms	100 ms	26.08 us
MessageECU5#24	ECU5	ECU3, ECU1	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU5#25	ECU5	ECU6, ECU3, ECU1	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU5#26	ECU5	ECU6, ECU3, ECU1, ECU0	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU5#27	ECU5	BROADCAST	3	IPv4+UDP	32.0	20 ms	20 ms	8.16 us
MessageECU5#28	ECU5	BROADCAST	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU5#29	ECU5	BROADCAST	3	IPv4+UDP	128.0	100 ms	100 ms	15.84 us
MessageECU6#0	ECU6	ECU1	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU6#1	ECU6	ECU3	3	IPv4+UDP	8.0	10 ms	10 ms	6.72 us
MessageECU6#2	ECU6	ECU1	3	IPv4+UDP	8.0	20 ms	20 ms	6.72 us
MessageECU6#3	ECU6	ECU3	3	IPv4+UDP	8.0	50 ms	50 ms	6.72 us
MessageECU6#4	ECU6	ECU1	3	IPv4+UDP	8.0	100 ms	100 ms	6.72 us
MessageECU6#5	ECU6	ECU3	3	IPv4+UDP	8.0	200 ms	200 ms	6.72 us
MessageECU6#6	ECU6	ECU1	3	IPv4+UDP	64.0	5 ms	5 ms	10.72 us
MessageECU6#7	ECU6	ECU3	3	IPv4+UDP	64.0	10 ms	10 ms	10.72 us
MessageECU6#8	ECU6	ECU1	3	IPv4+UDP	64.0	20 ms	20 ms	10.72 us
MessageECU6#9	ECU6	ECU3	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU6#10	ECU6	ECU1	3	IPv4+UDP	64.0	100 ms	100 ms	10.72 us
MessageECU6#11	ECU6	ECU3	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU6#12	ECU6	ECU1	3	IPv4+UDP	96.0	5 ms	5 ms	13.28 us
MessageECU6#13	ECU6	ECU3	3	IPv4+UDP	96.0	10 ms	10 ms	13.28 us
MessageECU6#14	ECU6	ECU1	3	IPv4+UDP	96.0	20 ms	20 ms	13.28 us
MessageECU6#15	ECU6	ECU3	3	IPv4+UDP	96.0	50 ms	50 ms	13.28 us
MessageECU6#16	ECU6	ECU1	3	IPv4+UDP	96.0	100 ms	100 ms	13.28 us
MessageECU6#17	ECU6	ECU3	3	IPv4+UDP	128.0	5 ms	5 ms	15.84 us
MessageECU6#18	ECU6	ECU1	3	IPv4+UDP	128.0	10 ms	10 ms	15.84 us
MessageECU6#19	ECU6	ECU3	3	IPv4+UDP	160.0	10 ms	10 ms	18.4 us
MessageECU6#20	ECU6	ECU1	3	IPv4+UDP	192.0	50 ms	50 ms	20.96 us
MessageECU6#21	ECU6	ECU3	3	IPv4+UDP	192.0	100 ms	100 ms	20.96 us
MessageECU6#22	ECU6	ECU1	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU6#23	ECU6	ECU3	3	IPv4+UDP	256.0	100 ms	100 ms	26.08 us
MessageECU6#24	ECU6	ECU3, ECU1	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU6#25	ECU6	ECU5, ECU3, ECU1	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU6#26	ECU6	ECU5, ECU3, ECU1, ECU0	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU6#27	ECU6	BROADCAST	3	IPv4+UDP	32.0	20 ms	20 ms	8.16 us
MessageECU6#28	ECU6	BROADCAST	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU6#29	ECU6	BROADCAST	3	IPv4+UDP	128.0	100 ms	100 ms	15.84 us
MessageECU7#0	ECU7	ECU1	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us
MessageECU7#1	ECU7	ECU3	3	IPv4+UDP	8.0	10 ms	10 ms	6.72 us
MessageECU7#2	ECU7	ECU1	3	IPv4+UDP	8.0	20 ms	20 ms	6.72 us
MessageECU7#3	ECU7	ECU3	3	IPv4+UDP	8.0	50 ms	50 ms	6.72 us
MessageECU7#4	ECU7	ECU1	3	IPv4+UDP	8.0	100 ms	100 ms	6.72 us
MessageECU7#5	ECU7	ECU3	3	IPv4+UDP	8.0	200 ms	200 ms	6.72 us
MessageECU7#6	ECU7	ECU1	3	IPv4+UDP	64.0	5 ms	5 ms	10.72 us
MessageECU7#7	ECU7	ECU3	3	IPv4+UDP	64.0	10 ms	10 ms	10.72 us
MessageECU7#8	ECU7	ECU1	3	IPv4+UDP	64.0	20 ms	20 ms	10.72 us
MessageECU7#9	ECU7	ECU3	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU7#10	ECU7	ECU1	3	IPv4+UDP	64.0	100 ms	100 ms	10.72 us
MessageECU7#11	ECU7	ECU3	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU7#12	ECU7	ECU1	3	IPv4+UDP	96.0	5 ms	5 ms	13.28 us
MessageECU7#13	ECU7	ECU3	3	IPv4+UDP	96.0	10 ms	10 ms	13.28 us
MessageECU7#14	ECU7	ECU1	3	IPv4+UDP	96.0	20 ms	20 ms	13.28 us
MessageECU7#15	ECU7	ECU3	3	IPv4+UDP	96.0	50 ms	50 ms	13.28 us
MessageECU7#16	ECU7	ECU1	3	IPv4+UDP	96.0	100 ms	100 ms	13.28 us
MessageECU7#17	ECU7	ECU3	3	IPv4+UDP	128.0	5 ms	5 ms	15.84 us
MessageECU7#18	ECU7	ECU1	3	IPv4+UDP	128.0	10 ms	10 ms	15.84 us
MessageECU7#19	ECU7	ECU3	3	IPv4+UDP	160.0	10 ms	10 ms	18.4 us
MessageECU7#20	ECU7	ECU1	3	IPv4+UDP	192.0	50 ms	50 ms	20.96 us
MessageECU7#21	ECU7	ECU3	3	IPv4+UDP	192.0	100 ms	100 ms	20.96 us
MessageECU7#22	ECU7	ECU1	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU7#23	ECU7	ECU3	3	IPv4+UDP	256.0	100 ms	100 ms	26.08 us
MessageECU7#24	ECU7	ECU3, ECU1	3	IPv4+UDP	8.0	5 ms	5 ms	6.72 us

Traffic Stream	Source	Destination(s)	PCP	Protocol	Payload	Period	Jitter	Dmin
MessageECU7#25	ECU7	ECU5, ECU3, ECU1	3	IPv4+UDP	64.0	200 ms	200 ms	10.72 us
MessageECU7#26	ECU7	ECU5, ECU3, ECU1, ECU0	3	IPv4+UDP	256.0	50 ms	50 ms	26.08 us
MessageECU7#27	ECU7	BROADCAST	3	IPv4+UDP	32.0	20 ms	20 ms	8.16 us
MessageECU7#28	ECU7	BROADCAST	3	IPv4+UDP	64.0	50 ms	50 ms	10.72 us
MessageECU7#29	ECU7	BROADCAST	3	IPv4+UDP	128.0	100 ms	100 ms	15.84 us
CAM#0	ECU1	ECU0	2	IPv4+UDP	1472.0	500 us	500 us	123.36 us
CAM#1	ECU3	ECU0	2	IPv4+UDP	1472.0	500 us	500 us	123.36 us
CAM#2	ECU5	ECU0	2	IPv4+UDP	1472.0	500 us	500 us	123.36 us
CAM#3	ECU7	ECU0	2	IPv4+UDP	1472.0	500 us	500 us	123.36 us
NRT#0	ECU0	BROADCAST	1	IPv4+UDP	1472.0	1 s	1 s	12.336 us
NRT#1	ECU7	BROADCAST	1	IPv4+UDP	1472.0	1 s	1 s	123.36 us
NRT#2	ECU2	BROADCAST	1	IPv4+UDP	1472.0	1 s	1 s	123.36 us
NRT#3	ECU4	BROADCAST	1	IPv4+UDP	1472.0	1 s	1 s	123.36 us

Table A.1: Traffic description

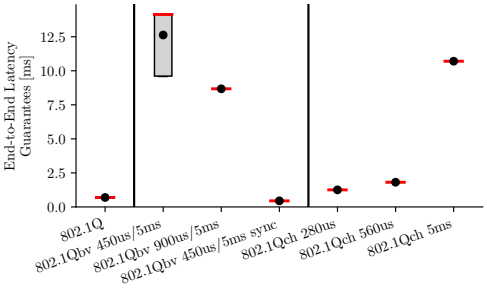
Appendix B

Additional Evaluation Results

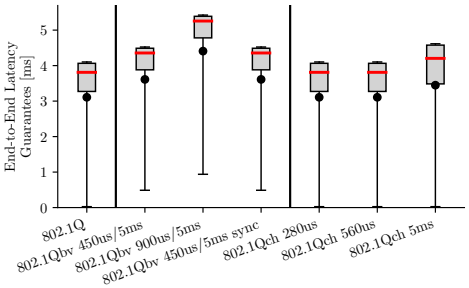
B.1 IEEE 802.1Q vs. IEEE 802.1Qbv and IEEE 802.1Qch

This section presents additional results for the double star, tree, and line topologies in Figures B.1, B.2, and B.3, respectively, accompanying the evaluation of Section 5.3.

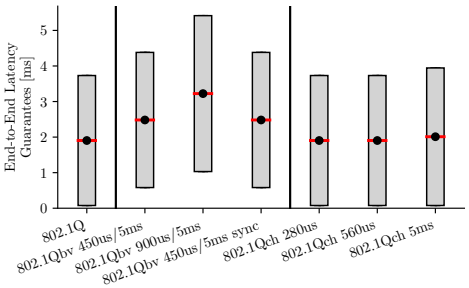
B.1.1 Double Star Topology



(a)



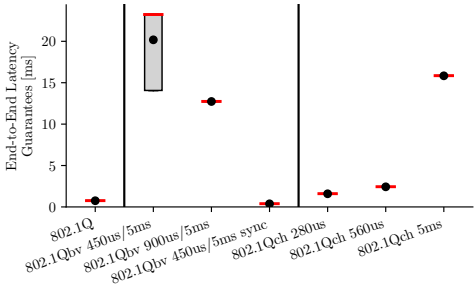
(b)



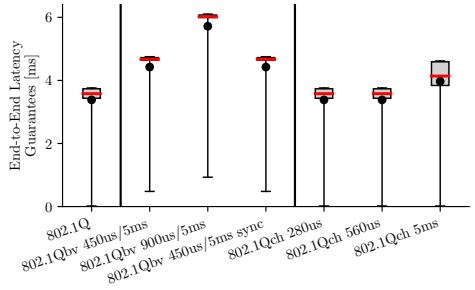
(c)

Figure B.1: End-to-end latency guarantee comparison for (a) PCP 4, (b) PCP 3, and (c) PCP 2 traffic in the double star topology

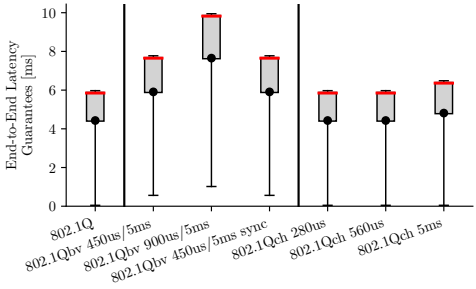
B.1.2 Tree Topology



(a)



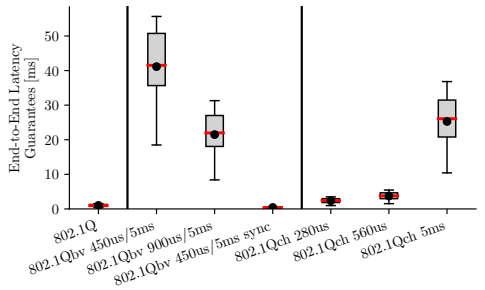
(b)



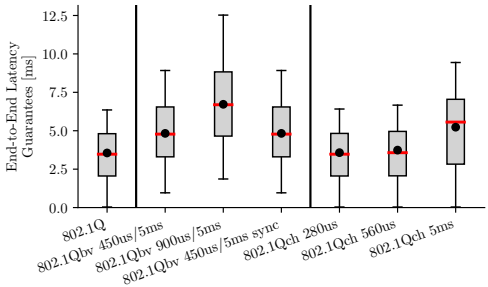
(c)

Figure B.2: End-to-end latency guarantee comparison for (a) PCP 4, (b) PCP 3, and (c) PCP 2 traffic in the tree topology

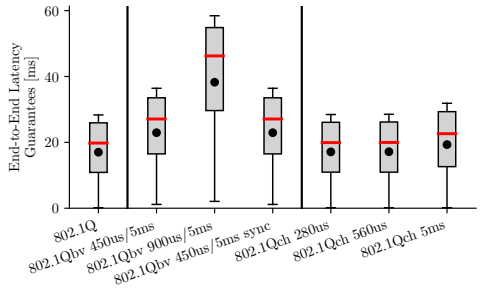
B.1.3 Line Topology



(a)



(b)



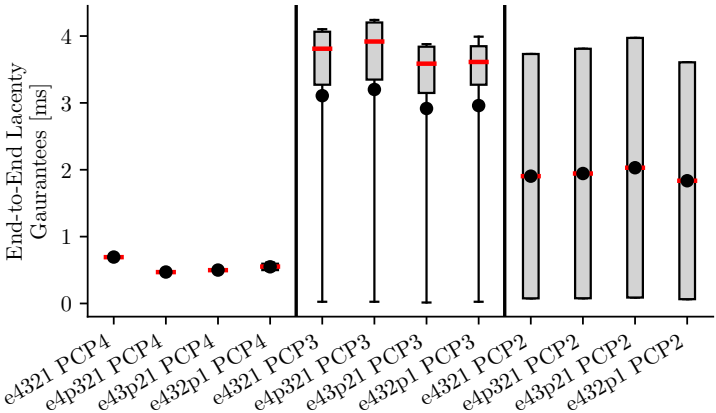
(c)

Figure B.3: End-to-end latency guarantee comparison for (a) PCP 4, (b) PCP 3, and (c) PCP 2 traffic in the line topology

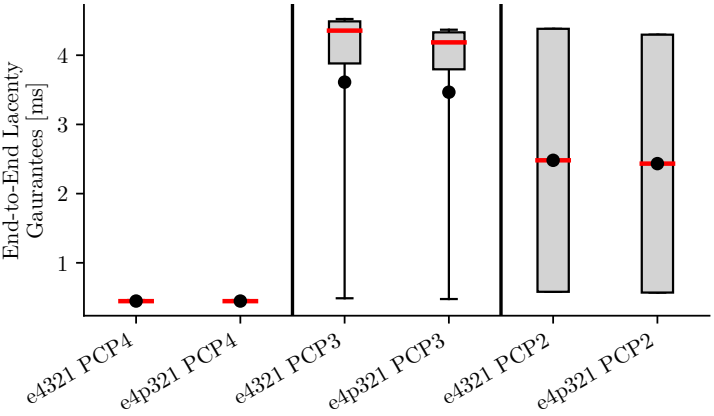
B.2 IEEE 802.1Q and IEEE 802.1Qbv with IEEE802.3br

This section presents additional results for the evaluation of Sections 5.4 and 5.5 for the double star, tree, and line topologies in Figures B.4, B.6, and B.8 for integration mode 1 and in Figures B.5, B.7, and B.9 for integration mode 2.

B.2.1 Double Star Topology in Integration Mode 1



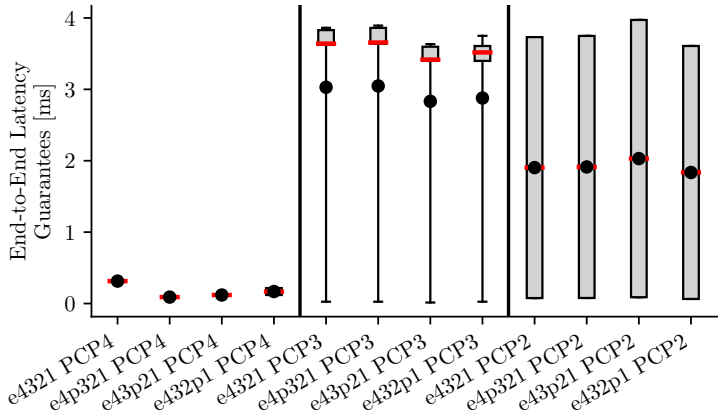
(a)



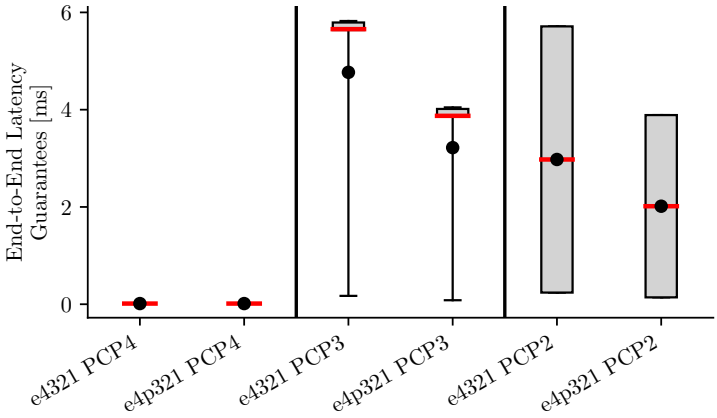
(b)

Figure B.4: End-to-end latency guarantee comparison with frame preemption for different traffic mappings in the double star topology and integration mode 1 for (a) IEEE 802.1Q and (b) IEEE 802.1Qbv.

B.2.2 Double Star Topology in Integration Mode 2



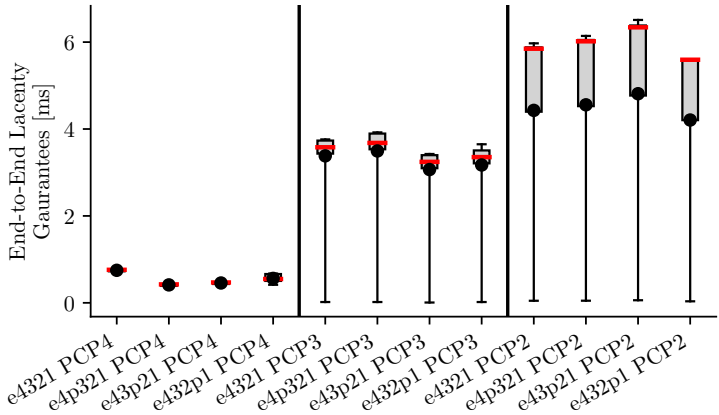
(a)



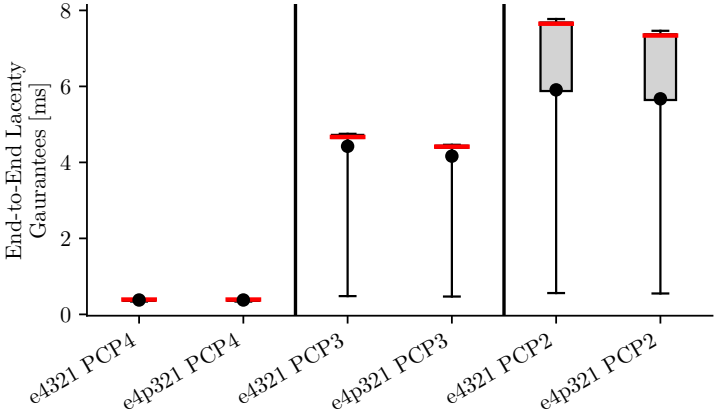
(b)

Figure B.5: End-to-end latency guarantee comparison with frame preemption for different traffic mappings in the double star topology and integration mode 2 for (a) IEEE 802.1Q and (b) IEEE 802.1Qbv.

B.2.3 Tree Topology in Integration Mode 1



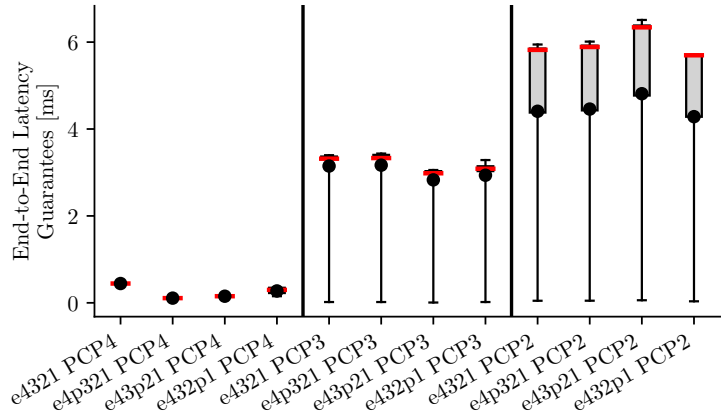
(a)



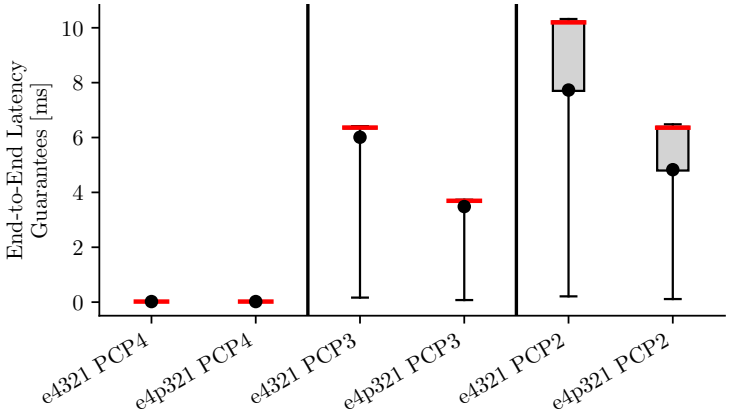
(b)

Figure B.6: End-to-end latency guarantee comparison with frame preemption for different traffic mappings in the tree topology and integration mode 1 for (a) IEEE 802.1Q and (b) IEEE 802.1Qbv.

B.2.4 Tree Topology in Integration Mode 2



(a)



(b)

Figure B.7: End-to-end latency guarantee comparison with frame preemption for different traffic mappings in the tree topology and integration mode 2 for (a) IEEE 802.1Q and (b) IEEE 802.1Qbv.

B.2.5 Line Topology in Integration Mode 1

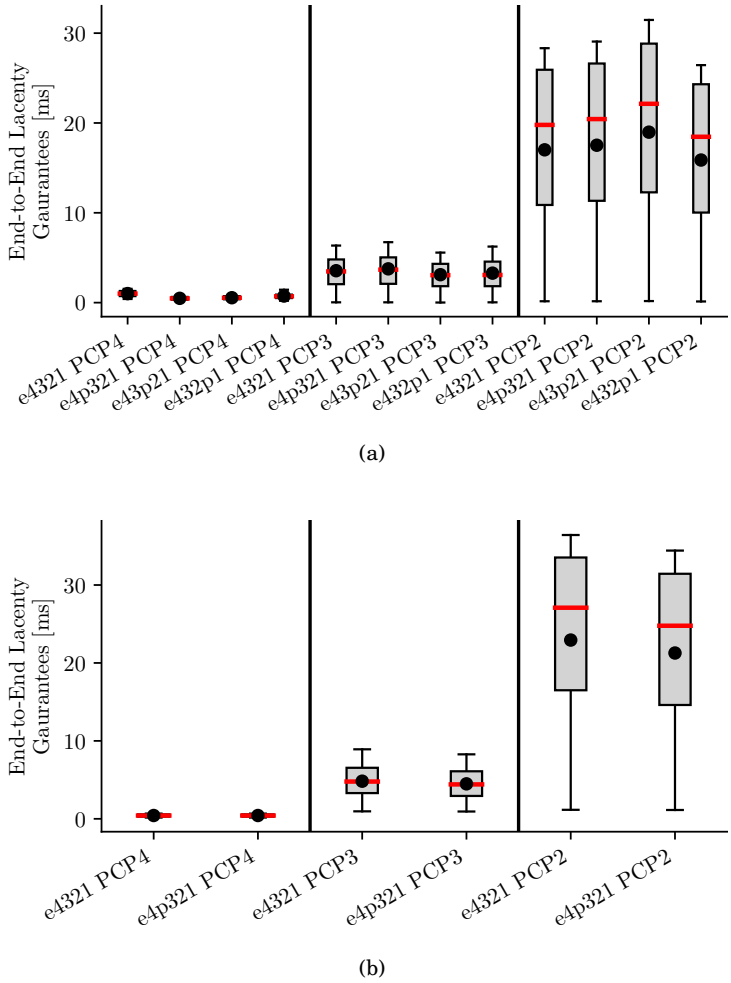
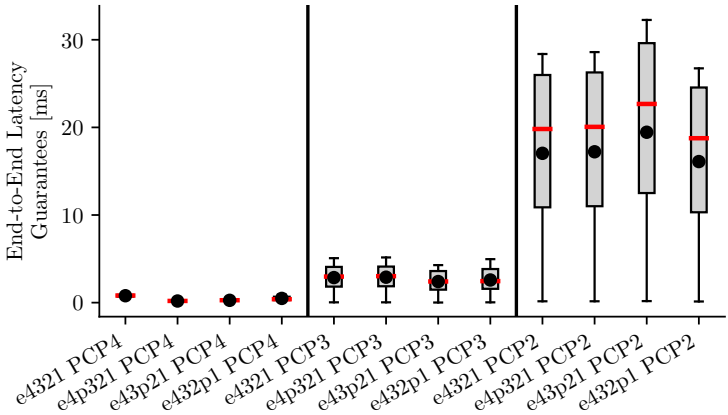
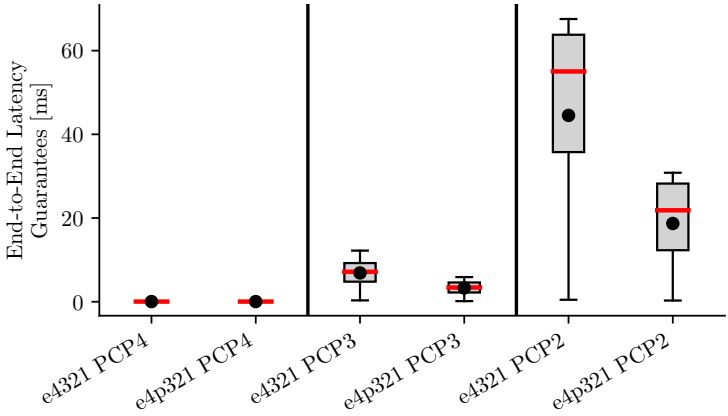


Figure B.8: End-to-end latency guarantee comparison with frame preemption for different traffic mappings in the line topology and integration mode 1 for (a) IEEE 802.1Q and (b) IEEE 802.1Qbv.

B.2.6 Line Topology in Integration Mode 2



(a)



(b)

Figure B.9: End-to-end latency guarantee comparison with frame preemption for different traffic mappings in the line topology and integration mode 2 for (a) IEEE 802.1Q and (b) IEEE 802.1Qbv.

Appendix C

Publications

C.1 Publications Related to this Thesis

C.1.1 Reviewed

Daniel Thiele and Rolf Ernst, "Formal Worst-Case Performance Analysis of Time-Sensitive Ethernet with Frame Preemption" in Proceedings of Emerging Technologies and Factory Automation (ETFA), (Berlin, Germany), pp. 9, September 2016.

This publication describes a compositional performance analysis for frame preemption in Ethernet according to IEEE 802.3br/IEEE 802.1Qbu. It compares frame preemption under priority-based transmission selection according to IEEE 802.1Q and Ethernet TSN's time-aware shaper according to IEEE 802.1Qbv. Section 4.6 is based on this publication. This publication received a best paper award.

Daniel Thiele and Rolf Ernst, "Formal Analysis Based Evaluation of Software Defined Networking for Time-Sensitive Ethernet" in Design Automation and Test in Europe (DATE), (Dresden, Germany), March 2016.

This publication evaluates the suitability of software defined networking (SDN) to automotive Ethernet using admission control as an illustrative example. It combines the worst-case communication latencies (in Ethernet) and the worst-case computation delays (at the SDN controller) via compositional performance analysis (underlining the generality of the compositional analysis framework).

Daniel Thiele and Rolf Ernst, "Formal Worst-Case Timing Analysis of Ethernet TSN's Burst-Limiting Shaper" in Design Automation and Test in Europe (DATE), (Dresden, Germany), March 2016.

This publication describes a compositional analysis based approach for the burst-limiting shaper, that was under consideration for standardization within Ethernet TSN at that time. The burst-limiting shaper has never been standardized.

Daniel Thiele, Rolf Ernst and Jonas Diemer, "Formal Worst-Case Timing Analysis of Ethernet TSN's Time-Aware and Peristaltic Shapers" in Vehicular Networking Conference (VNC), (Kyoto, Japan), December 2015.

This publication presents a compositional performance analysis for Ethernet TSN's time-aware shaper (IEEE 802.1Qbv) and peristaltic shaper (IEEE 802.1Qch) and compares them to priority-based transmission selection according to IEEE 802.1Q. Sections 4.4 and 4.5 are based on this publication.

Daniel Thiele, Philip Axer and Rolf Ernst, "Improving Formal Timing Analysis of Switched Ethernet by Exploiting FIFO Scheduling" in Design Automation Conference (DAC), (San Francisco, CA, USA), June 2015.

This publication proposes a compositional performance analysis to exploit FIFO correlations between consecutive switch ports in IEEE 802.1Q to derive tighter worst-case guarantees.

Daniel Thiele, Philip Axer, Rolf Ernst and Jan R. Seyler, "Improving Formal Timing Analysis of Switched Ethernet by Exploiting Traffic Stream Correlations" in Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), (New Delhi, India), October 2014.

This publication describes an algorithm to exploit workload correlations between traffic streams during formal analysis by exploiting that frames, depending on their length, have a certain distance between their arrival times. The general workload correlation described in Section 4.7 covers these cases and leads to a more efficient implementation.

Daniel Thiele, Jonas Diemer, Philip Axer, Rolf Ernst and Jan Seyler, "Improved Formal Worst-Case Timing Analysis of Weighted Round Robin Scheduling for Ethernet" in In Proc. of CODES+ISSS, (Montreal, Canada), September 2013.

This publication presents a compositional performance analysis for weighted round robin based transmission selection in Ethernet. In contrast to related work on formal analysis of weighted round robin, this publication takes the actual interfering load into account.

C.1.2 Unreviewed

Haibo Zeng, Prachi Joshi, Daniel Thiele, Jonas Diemer, Philip Axer, Rolf Ernst and Petru Eles, "Handbook of Hardware/Software Codesign - Chapter: Networked Real-Time Embedded Systems"Soonhoi Ha and Jürgen Teich, Ed., Springer, 2017.

This contribution to a book chapter summarizes the CPA-based formal analysis approach for automotive Ethernet networks using priority-based transmission selection according to IEEE 802.1Q as an illustrative example.

Mischa Moestl, Daniel Thiele and Rolf Ernst, "Towards Fail-operational Ethernet Based In-vehicle Networks" in Proceedings of the 53rd Annual Design Automation Conference, ACM, 2016

This invited publication discusses how fail-operational Ethernet-based networks could be conceived. It covers switch and protocol considerations as well as the mitigation of transient and permanent faults.

Daniel Thiele, Philip Axer, Rolf Ernst, Jonas Diemer and Kai Richter, "Cooperating on Real-Time Capable Ethernet Architecture in Vehicles" in Proc. of VDI Internationaler Kongress Elektronik im Fahrzeug, Baden-Baden, October 2013.

This article looks at the (then) current state of automotive Ethernet from a topology, switch architecture, and communication protocol point of view and calls upon standardized approaches.

Daniel Thiele, Philip Axer, Rolf Ernst, Jonas Diemer and Kai Richter, "Cooperating on Real-Time Capable Ethernet Architecture in Vehicles (revised version of VDI conference paper)", ATZelektronik worldwide, Springer, 2013, revised version of VDI conference paper.

This publication is a revision of "Cooperating on Real-Time Capable Ethernet Architecture in Vehicles" for publication in the ATZelektronik magazine.

C.2 Publications Unrelated to this Thesis

C.2.1 Reviewed

Daniel Thiele, Johannes Schlatow, Philip Axer and Rolf Ernst, "Formal timing analysis of CAN-to-Ethernet gateway strategies in automotive networks, Real-Time Systems, 2015.

This journal article identifies CAN-to-Ethernet timing analysis as one of the important use cases of automotive (backbone) Ethernet networks. Due to the significantly higher protocol overhead of Ethernet (compared to CAN), multiplexing of CAN frames into Ethernet frames is required. The article proposes formal methods to evaluate different multiplexing strategies, including both their impact on Ethernet as well as on the CAN-to-Ethernet gateway.

Philip Axer, Daniel Thiele, Rolf Ernst and Jonas Diemer, "Exploiting Shaper Context to Improve Performance Bounds of Ethernet AVB Networks" in Proc. of DAC, (San Francisco, USA), June 2014.

This publication describes a formal analysis that exploits the fact that, on an Ethernet link, the bandwidth is limited to the minimum of the link speed and the context of the Ethernet AVB traffic shaper. Section 4.7 of this thesis generalizes this analysis to arbitrary heterogeneous link speeds. In this thesis, this approach has been implemented for the formal analyses for IEEE 802.1Q (priority-based transmission selection) and IEEE 802.1Qbv.

Philip Axer, Daniel Thiele and Rolf Ernst, "Formal Timing Analysis of Automatic Repeat Request for Switched Real-Time Networks" in In Proc. of SIES, (Pisa, Italy), June 2014.

This publication presents a formal analysis of error control protocols (namely stop-and-wait and go-back-N) for real-time Ethernet communication.

Philip Axer, Daniel Thiele, Rolf Ernst, Jonas Diemer, Simon Schliecker and Kai Richter, "Requirements on Real-Time-Capable Automotive Ethernet Architectures" in Proc. of SAE World Congress, 2014.

This publication presets the requirements for Ethernet as an automotive communication backbone. It proposes formal methods as well as metrics for design space exploration. The argumentation is supported by the application of said methods and metrics to actual automotive Ethernet topologies and transmission selection mechanisms.

Henning Sahlbach, Daniel Thiele and Rolf Ernst, "A System-Level FPGA Design Methodology for Video Applications with Weakly-Programmable Hardware Components", Journal of Real-Time Image Processing, Springer Berlin Heidelberg, March 2014.

This publication describes a concept for system-level FPGA development targeting video applications with weakly-programmable hardware components, combining rapid software prototyping, component-based FPGA design, and formal real-time analysis methods.

Jonas Diemer, Daniel Thiele and Rolf Ernst, "Formal Worst-Case Timing Analysis of Ethernet Topologies with Strict-Priority and AVB Switching" in 7th IEEE International Symposium on Industrial Embedded Systems (SIES12), Juni 2012, Invited Paper.

This publication describes an analysis for Ethernet AVB in the compositional performance analysis framework. It compares strict priority based transmission selection with AVB's credit-based traffic shaper for different topologies.

Daniel Thiele and Rolf Ernst, "Optimizing Performance Analysis for Synchronous Dataflow Graphs with Shared Resources" in Proc. of Design, Automation, and Test in Europe (DATE), (Dresden, Germany), March 2012.

This publication presents a formal approach to integrate resource sharing into SDF graphs to derive deadlines for shared resource accesses and memory requirements.

Bibliography

[1] pyCPA. https://bitbucket.org/pycpa/. Accessed: 2016-07-01.

[2] 802.3 WG - Ethernet Working Group. 802.3bp-2016 - IEEE Standard for Ethernet Amendment 4: Physical Layer Specifications and Management Parameters for 1 Gb/s Operation over a Single Twisted-Pair Copper Cable. https://standards.ieee.org/standard/802_3bp-2016.html.

[3] 802.3 WG - Ethernet Working Group. 802.3br-2016 - IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic. https://standards.ieee.org/standard/802_3br-2016.html.

[4] 802.3 WG - Ethernet Working Group. 802.3bw-2015 - 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). https://standards.ieee.org/standard/802_3bw-2015.html.

[5] 802.3 WG - Ethernet Working Group. IEEE 802.3cg-2019 - IEEE Standard for Ethernet - Amendment 5: Physical Layer Specifications and Management Parameters for 10 Mb/s Operation and Associated Power Delivery over a Single Balanced Pair of Conductors. https://standards.ieee.org/standard/802_3cg-2019.html, 2019.

[6] ARINC Industry Activities. *664P7-1 Aircraft Data Network, Part 7, Avionics Full-Duplex Switched Ethernet Network (AFDX)*. 2009.

[7] Leonie Ahrendts, Robin Hofmann, and Rolf Ernst. Cpa - compositional performance analysis. In Soonhoi Ha and Jürgen Teich, editors, *Handbook of Hardware/Software Codesign*, chapter 23. Springer-Verlag, Berlin, Heidelberg, 2017.

[8] Giuliana Alderisi, Alfio Caltabiano, Giancarlo Vasta, Giancarlo Iannizzotto, Till Steinbach, and Lucia Lo Bello. Simulative Assessments of IEEE 802.1 Ethernet AVB and Time-Triggered Ethernet for Advanced Driver Assistance Systems and In-car Infotainment. In *IEEE Vehicular Networking Conference (VNC)*, Seoul, South Korea, November 2012.

[9] Giuliana Alderisi, Gaetano Patti, and Lucia Lo Bello. Introducing support for scheduled traffic over IEEE audio video bridging networks. In *Proceedings of IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–9, Cagliari, Italy, September 2013.

[10] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681, RFC Editor, September 2009.

[11] Anaïs Finzi and Ahlem Mifdaoui and Fabrice Frances and Emmanuel Lochin. Incorporating TSN/BLS in AFDX for mixed-criticality applications: Model and timing analysis. In *14th IEEE International Workshop on Factory Communication Systems, WFCS 2018, Imperia, Italy, June 13-15, 2018*, pages 1–10, 2018.

[12] Anaïs Finzi and Ahlem Mifdaoui and Fabrice Frances and Emmanuel Lochin. Network Calculus-based Timing Analysis of AFDX networks with Strict Priority and TSN/BLS Shapers. In *13th IEEE International Symposium on Industrial Embedded Systems, SIES 2018, Graz, Austria, June 6-8, 2018*, pages 1–10, 2018.

[13] Ashjaei, Mohammad and Patti, Gaetano and Behnam, Moris and Nolte, Thomas and Alderisi, Giuliana and Lo Bello, Lucia. Schedulability analysis of Ethernet Audio Video Bridging networks with scheduled traffic support. *Real-Time Systems*, 53(4):526–577, Jul 2017.

[14] Avionic Systems Group AS-2D2. TTEthernet (SAE AS6802). http://standards.sae.org/as6802/.

[15] Philip Axer. *Performance of Time-Critical Embedded Systems under the Influence of Errors and Error Handling Protocols*. Ph.d. thesis, Technische Universität Braunschweig, Institute of Computer and Network Engineering, 2016.

[16] Philip Axer, Daniel Thiele, and Rolf Ernst. Formal Timing Analysis of Automatic Repeat Request for Switched Real-Time Networks. In *In Proc. of SIES*, Pisa, Italy, June 2014.

[17] Philip Axer, Daniel Thiele, Rolf Ernst, and Jonas Diemer. Exploiting Shaper Context to Improve Performance Bounds of Ethernet AVB Networks. In *Proceedings of DAC*, San Francisco, 2014.

[18] Philip Axer, Daniel Thiele, Rolf Ernst, Jonas Diemer, Simon Schliecker, and Kai Richter. Requirements on Real-Time-Capable Automotive Ethernet Architectures. In *Proc. of SAE World Congress*, 2014. doi:10.4271/2014-01-0245.

[19] H. Ayed, A. Mifdaoui, and C. Fraboul. Hierarchical traffic shaping and frame packing to reduce bandwidth utilization in the afdx. In *Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014)*, pages 77–86, 2014.

[20] B. Cain and S. Deering and I. Kouvelas and B. Fenner and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376, RFC Editor, October 2002.

[21] B. Cain and S. Deering and I. Kouvelas and B. Fenner and A. Thyagarajan. Internet Group Management Protocol, Version 3. RFC 3376, RFC Editor, October 2002.

[22] Iain John Bate. *Scheduling and Timing Analysis for Safety Critical Real-Time Systems*. Ph.d. thesis, University of York, Department of Computer Science, 1998.

[23] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network. In *IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, 2009.

[24] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Improving the Worst-Case Delay Analysis of an AFDX Network Using an Optimized Trajectory Approach. *IEEE Transactions on Industrial Informatics*, 2010.

[25] H. Bauer, J.-L. Scharbarg, and C. Fraboul. Worst-Case Backlog Evaluation of Avionics Switched Ethernet Networks with the Trajectory Approach. In *Euromicro Conference on Real-Time Systems (ECRTS)*, 2012.

[26] J.C.R. Bennett and Hui Zhang. Hierarchical packet fair queueing algorithms. *Networking, IEEE/ACM Transactions on*, 5(5):675–689, oct 1997.

[27] Unmesh D. Bordoloi, Amir Aminifar, Petru Eles, and Zebo Peng. Schedulability Analysis of Ethernet AVB Switches. In *International Conference on Real-Time Computing Systems and Applications (RTCSA)*, 2014.

[28] Julian Brand. Dynamic In-vehicle Network Management. In *Automotive Ethernet Congress*, 2019.

[29] Broadcom Corporation. BroadR-Reach Physical Layer Transceiver Specification For Automotive Applications, V3.0.

[30] Broadcom Limited. BCM89500/BCM89501 Automotive Switching Technolog - 7-Port Integrated BroadR-Reach Automotive Switch. https://www.broadcom.com/collateral/pb/89500_89501-PB00-R.pdf, 2011.

[31] Broadcom Limited. ROBOSwitch Overview. https://community.broadcom.com/servlet/JiveServlet/previewBody/1575-102-4-1598/Roboswitch_v2.pdf, 2014.

[32] Cisco Systems Inc. Cisco Nexus 7000 F2-Series 48-Port 1 and 10 Gigabit Ethernet Module Data Sheet. Technical report.

[33] Cisco Systems Inc. Increase Flexibility in Layer 2 Switches by Integrating Ethernet ASSP Functions Into FPGAs. White Paper WP-AAB091005-1.1, 2006.

[34] Cisco Systems Inc. Cisco Nexus 3064PQ Switch Architectur. White Paper, 2011.

[35] European Commission. Road safety: Commission welcomes agreement on new EU rules to help save lives. March 2019.

[36] International Electrotechnical Commission. *IEC 61158*. 2003.

[37] AUTOSAR Development Cooperation. AUTOSAR Release 4.2.2 - Specification of Service Discovery. Document Identification No 616, July 2015.

[38] AUTOSAR Development Cooperation. AUTOSAR Release 4.3.1 - Specification of Ethernet Switch Driver. Document Identification No 656, December 2017.

[39] Érika Cota, Alexandre de Morais Amory, and Marcelo Soares Lubaszewski. *Reliability, Availability and Serviceability of Networks-on-Chip*. Springer US, 2012.

[40] Silviu Craciunas, Ramon Serna Oliver, Martin Chmelik, and Wilfried Steiner. Scheduling real-time communication in ieee 802.1qbv time sensitive networks. 10 2016.

[41] Robert I. Davis and Alan Burns. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Systems*, 35, 2007.

[42] Stephen E. Deering and Robert M. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, RFC Editor, December 1998.

[43] L. Delgrossi and L. Berger. Internet Stream Protocol Version 2 (ST2) - Protocol Specification - Version ST2+. RFC 1819, RFC Editor, August 1995.

[44] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Symposium proceedings on Communications architectures & protocols*, SIGCOMM '89, pages 1–12, New York, NY, USA, 1989. ACM.

[45] Jonas Diemer. *Predictable Architectures and Performance Analysis for General-Purpose Networks-on-Chip*. Ph.d. thesis, Technische Universität Braunschweig, Institute of Computer and Network Engineering, 2016.

[46] Jonas Diemer, Philip Axer, and Rolf Ernst. Compositional Performance Analysis in Python with pyCPA. In *Int. Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2012.

[47] Jonas Diemer, Jonas Rox, and Rolf Ernst. Modeling of Ethernet AVB Networks for Worst-Case Timing Analysis. In *MATHMOD - Vienna International Conference on Mathematical Modelling*, Vienna, Austria, 2012.

[48] Jonas Diemer, Jonas Rox, Rolf Ernst, Feng Chen, Karl-Theo Kremer, and Kai Richter. Exploring the Worst-Case Timing of Ethernet AVB for Industrial Applications. In *Proc. of the 38th Annual Conference of the IEEE Industrial Electronics Society*, Montreal, Canada, October 2012.

[49] Jonas Diemer, Jonas Rox, Mircea Negrean, Steffen Stein, and Rolf Ernst. Real-time communication analysis for networks with two-stage arbitration. In *Proceedings of the ninth ACM International Conference on Embedded Software (EMSOFT)*, EMSOFT 2011, pages 243–252, Taipei, Taiwan, oct 2011. ACM.

[50] Jonas Diemer, Daniel Thiele, and Rolf Ernst. Formal Worst-Case Timing Analysis of Ethernet Topologies with Strict-Priority and AVB Switching. In *IEEE International Symposium on Industrial Embedded Systems*, 2012.

[51] Frank Dürr and Naresh Ganesh Nayak. 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, pages 203–212, New York, NY, USA, 2016. Association for Computing Machinery.

[52] E Wandeler. *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. Ph.d. thesis, ETH Zürich, 2006.

[53] F. Frances and C. Fraboul and F. Grieu. Using network calculus to optimize the AFDX network. In *Embedded Real-Time Software and Systems (ERTS)*, Jan. 2006.

[54] F. Smirnov and M. Glaß and F. Reimann and J. Teich. Formal timing analysis of non-scheduled traffic in automotive scheduled TSN networks. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1643–1646, March 2017.

[55] A. Farrel, J.-P. Vasseur, and J. Ash. A Path Computation Element (PCE)-Based Architecture. RFC 4655, RFC Editor, August 2006.

[56] N. Feiertag, K. Richter, J. Nordlander, and J. Jonsson. A Compositional Framework for End-to-End Path Delay Calculation of Automotive Systems under Different Path Semantics. In *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2008.

[57] A. Finzi and A. Mifdaoui. Worst-case timing analysis of afdx networks with multiple tsn/bls shapers. *IEEE Access*, 8:106765–106784, 2020.

[58] R. T. Flores and M. Boyer. Performance analysis of the Disrupted Static Priority scheduling for AFDX. In *Formal Methods and Models for Codesign (MEMOCODE)*, pages 94–103, Oct 2014.

[59] Simon Fürst. System/ software architecture for autonomous driving systems. In *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 31–32, March 2019.

[60] G. Kemayo and N. Benammar and F. Ridouard and H. Bauer and P. Richard. Improving AFDX end-to-end delays analysis. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sep. 2015.

[61] Voica Gavrilut and Paul Pop. Scheduling in time sensitive networks (tsn) for mixed-criticality industrial applications. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 1–4, 2018.

[62] Voica Gavrilut and Paul Pop. Traffic-type assignment for tsn-based mixed-criticality cyber-physical systems. *Acm Transactions on Cyber-physical Systems*, 4(2):23, 2020.

[63] Voica Gavrilut, Luxi Zhao, Michael L. Raagaard, and Paul Pop. Avb-aware routing and scheduling of time-triggered traffic for tsn. *IEEE Access*, 6:75229–75243, 2018.

[64] J.J. Gutiérrez, J.C. Palencia, and M. González Harbour. Response Time Analysis in AFDX Networks. In *XIV Jornadas de Tiempo Real*, February 2011.

[65] David Hellmanns, Jonathan Falk, Alexander Glavackij, Rene Hummen, Stephan Kehrer, and Frank Dürr. On the performance of stream-based, class-based time-aware shaping and frame preemption in tsn. In *2020 IEEE International Conference on Industrial Technology (ICIT)*, pages 298–303, 2020.

[66] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System Level Performance Analysis - the SymTA/S Approach. In *IEE Proceedings Computers and Digital Techniques*, 2005.

[67] Thomas Hogenmüller. 1 Twisted Pair 100 Mbit/s Ethernet (1TPCE) - Call for Interest at IEEE802.3 Working Group. In *IEEE 802.3 Plenary Meeting*, Beijing, March 2014.

[68] Hyung-Taek Lim. Real-Time Communication in an IP/Ethernet-based In-car Network, November 2011.

[69] IEEE Audio Video Bridging Task Group. 802.1AS - Timing and Synchronization. http://www.ieee802.org/1/pages/802.1as.html.

[70] IEEE Audio Video Bridging Task Group. 802.1BA - Audio Video Bridging (AVB) Systems. http://www.ieee802.org/1/pages/802.1ba.html.

[71] IEEE Audio Video Bridging Task Group. 802.1Qat - Stream Reservation Protocol. http://www.ieee802.org/1/pages/802.1at.html. superseded by IEEE 802.1Q.

[72] IEEE Audio Video Bridging Task Group. 802.1Qav - Forwarding and Queuing Enhancements for Time-Sensitive Streams. http://www.ieee802.org/1/pages/802.1av.html. superseded by IEEE 802.1Q.

[73] IEEE Computer Society LAN/MAN Standards Committee. IEEE 802.3ab-1999 - 1000BASE-T. http://www.ieee802.org/3/ab, June 1999.

[74] IEEE Computer Society LAN/MAN Standards Committee. IEEE 802.3-2012 - IEEE Standard for Ethernet, 2012.

[75] IEEE Instrumentation and Measurement Society . IEEE 1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, 2008.

[76] IEEE Standards Association. IEEE 802.1Q - Bridges and Bridged Networks, 2014.

[77] IEEE Time-Sensitive Networking Task Group. 802.1AS-Rev - Timing and Synchronization for Time-Sensitive Applications. http://www.ieee802.org/1/pages/802.1AS-rev.html.

[78] IEEE Time-Sensitive Networking Task Group. 802.1CB - Frame Replication and Elimination for Reliability. http://www.ieee802.org/1/pages/802.1cb.html.

[79] IEEE Time-Sensitive Networking Task Group. 802.1Qbu - Frame Preemption. http://www.ieee802.org/1/pages/802.1bu.html.

[80] IEEE Time-Sensitive Networking Task Group. 802.1Qch - Cyclic Queuing and Forwarding. http://www.ieee802.org/1/pages/802.1ch.html.

[81] IEEE Time-Sensitive Networking Task Group. 802.1Qci - Per-Stream Filtering and Policing. http://www.ieee802.org/1/pages/802.1ci.html.

[82] IEEE Time-Sensitive Networking Task Group. 802.1Qcr - Bridges and Bridged Networks Amendment: Asynchronous Traffic Shaping. http://www.ieee802.org/1/pages/802.1cr.html.

[83] IEEE Time-Sensitive Networking Task Group. IEEE 802.1Qbv - Enhancements for Scheduled Traffic. https://standards.ieee.org/findstds/standard/802.1Qbv-2015.html.

[84] IEEE Time-Sensitive Networking Task Group. IEEE P802.1Qbb - Priority-based Flow Control. http://www.ieee802.org/1/pages/802.1bb.html.

[85] IEEE Time-Sensitive Networking Task Group. IEEE Time-Sensitive Networking Task Group. http://www.ieee802.org/1/pages/tsn.html.

[86] IEEE Time-Sensitive Networking Task Group. P802.1Qca (Draft 2.1) - Path Control and Reservation. http://www.ieee802.org/1/pages/802.1ca.html.

[87] IEEE Time-Sensitive Networking Task Group. P802.1Qcc (Draft 0.4) - Stream Reservation Protocol (SRP) Enhancements and Performance Improvements. http://www.ieee802.org/1/pages/802.1cc.html.

[88] J. Imtiaz, J. Jasperneite, and L. Han. A performance study of Ethernet Audio Video Bridging (AVB) for Industrial real-time communication. In *IEEE Conference on Emerging Technologies Factory Automation*, 2009.

[89] Information Sciences Institute University of Southern California. Internet Protocol. RFC 791, RFC Editor, September 1981.

[90] Information Sciences Institute University of Southern California. Transmission Protocol. RFC 793, RFC Editor, September 1981.

[91] International Electrotechnical Commission. IEC 61508:2010 - Functional Safety of Electrical/Electronic/Programmable Safety-related Systems, 2010.

[92] International Organization for Standardization (ISO). *ISO/IEC 7498-1 - Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*. 1994.

[93] International Organization for Standardization (ISO). ISO 26262: Road Vehicles - Functional Safety, 2011.

[94] International Organization for Standardization (ISO). *ISO 17458 - Road vehicles - FlexRay communications system*. 2013.

[95] International Telecommunication Union - Telecommunication Standardization Sector (ITU-T). *Recommendation I.150 - B-ISDN asynchronous transfer mode functional characteristics*. 1999.

[96] J. Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, RFC Editor, January 1984.

[97] J. Scharbarg and C. Fraboul. Simulation for end-to-end delays distribution on a switched Ethernet. In *IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*, pages 1092–1099, Sep. 2007.

[98] Wen-Kang Jia, Gen-Hen Liu, and Yaw-Chung Chen. Performance Evaluation of IEEE 802.1Qbu: Experimental and Simulation Results. In *IEEE Conference on Local Computer Networks (LCN)*, 2013.

[99] Bengt Jonsson, Simon Perathoner, Lothar Thiele, and Wang yi. Cyclic dependencies in modular performance analysis. pages 179–188, 01 2008.

[100] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *BCS Computer Journal*, 25(5):390–395, 1986.

[101] M. Karol, M. Hluchyj, and S. Morgan. Input versus output queueing on a space-division packet switch. *IEEE Transactions on Communications*, 35(12):1347–1356, Dec 1987.

[102] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis. Weighted round-robin cell multiplexing in a general-purpose ATM switch chip. *Selected Areas in Communications, IEEE Journal on*, 9(8), oct 1991.

[103] G. Kemayo, F. Ridouard, H. Bauer, and P. Richard. Optimistic problems in the trajectory approach in FIFO context. In *IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, 2013.

[104] Kemayo, Georges and Ridouard, Frédéric and Bauer, Henri and Richard, Pascal. A forward end-to-end delays analysis for packet switched networks. In *Proceedings of the 22Nd International Conference on Real-Time Networks and Systems*, RTNS '14, pages 65:65–65:74, New York, NY, USA, 2014. ACM.

[105] Sven Kerschbaum, Franz-Josef Götz, and Feng Chen. Towards the Calculation of Performance Guarantees for BLS in Time-Sensitive Networks. In *IEEE 802.1 TSN TG Meeting*, Dallas, TX, USA, November 2013.

[106] Joonkyo Kim, Bum Yong Lee, and Jaehyun Park. Preemptive switched ethernet for real-time process control system. In *IEEE Conference on Industrial Informatics (INDIN)*, 2013.

[107] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1), Jan 2015.

[108] L. Zhao and P. Pop and S. S. Craciunas. Worst-Case Latency Analysis for IEEE 802.1Qbv Time Sensitive Networks Using Network Calculus. *IEEE Access*, 6:41803–41815, 2018.

[109] L. Zhao and P. Pop and Z. Zheng and Q. Li. Timing Analysis of AVB Traffic in TSN Networks Using Network Calculus. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 25–36, April 2018.

[110] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE P802.1DG - Time-Sensitive Networking Profile for Automotive In-Vehicle Ethernet Communications. https://1.ieee802.org/tsn/802-1dg/.

[111] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE 803.x-1997 - IEEE Standard for Local and Metropolitan Area Networks: Specification for 802.3 Full Duplex Operation, 1997. superseded by IEEE 802.3.

[112] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE 802.1aq - Shortest Path Bridging, 2012.

[113] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE 802-2014 - IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture, 2014.

[114] LAN/MAN Standards Committee of the IEEE Computer Society. IEEE 802.1AX-2014 - Link Aggregation, 2014.

[115] Jean-Yves Le Boudec. A theory of traffic regulators for deterministic networks with application to interleaved regulators. *IEEE/ACM Transactions on Networking*, 26(6):2721–2733, 2018.

[116] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: A Theory of Deterministic Queuing Systems for the Internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[117] Kyung Chang Lee, Suk Lee, and Man Hyung Lee. Worst Case Communication Delay of Real-Time Industrial Switched Ethernet With Multiple Levels. *IEEE Transactions on Industrial Electronics*, oct. 2006.

[118] J. P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th Real-Time Systems Symposium*, pages 201–209, 1990.

[119] Xiaoting Li, Oliver Cros, and Laurent George. The Trajectory approach for AFDX FIFO networks revisited and corrected. In *RTCSA*, 2014.

[120] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1):46–61, 1973.

[121] Lakshmi Mandyam and B. Kinney. Switch Fabric Implementation Using Shared Memory. Application Note 1704, 2014.

[122] S. Martin and P. Minet. Schedulability Analysis of Flows Scheduled with FIFO: Application to the Expedited Forwarding Class. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[123] Kirsten Matheus and Thomas Königseder. *Automotive Ethernet*. Cambridge University Press, 2017.

[124] Robert McGill, John W. Tukey, and Wayne A. Larsen. Variations of Box Plots. *The American Statistician*, 32(1):12–16, February 1978.

[125] Nick McKeown. The iSLIP Scheduling Algorithm for Input-queued Switches. *IEEE/ACM Transactions on Networking*, 7(2):188–201, April 1999.

[126] Shane D. McLean, Emil Alexander Juul Hansen, Paul Pop, and Silviu S. Craciunas. Configuring adas platforms for automotive applications using metaheuristics. *Frontiers in Robotics and AI*, 8, 2022.

[127] P. Meyer, T. Steinbach, F. Korf, and T.C. Schmidt. Extending IEEE 802.1 AVB with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic. In *Vehicular Networking Conference (VNC), 2013 IEEE*, pages 47–54, Dec 2013.

[128] Ahlem Mifdaoui and Thierry Leydier. Beyond the Accuracy-Complexity Tradeoffs of CompositionalAnalyses using Network Calculus for Complex Networks. In *10th International Workshop on Compositional Theory and Technology for Real-Time Embedded Systems (co-located with RTSS 2017)*, pages pp. 1–8, Paris, France, December 2017.

[129] V. Mikolasek, A. Ademaj, and S. Racek. Segmentation of Standard Ethernet Messages in the Time-Triggered Ethernet. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2008.

[130] Ehsan Mohammadpour, Eleni Stai, Maaz Mohiuddin, and Jean-Yves Le Boudec. Latency and backlog bounds in time-sensitive networking with credit based shapers and asynchronous traffic shaping. In *2018 30th International Teletraffic Congress (ITC 30)*, volume 02, pages 1–6, 2018.

[131] Mischa Möstl, Daniel Thiele, and Rolf Ernst. Towards Fail-Operational Ethernet Based In-Vehicle Networks. In *Design Automation Conference (DAC)*, Autin, TX, USA, June 2016. invited paper, to appear.

[132] Nicolas Navet, Jan Seyler, and Jörn Migge. Timing verication of real-time automotive Ethernet networks: what can we expect from simulation? In *European Congress on Embedded Real Time Software and Systems (ERTS^2)*, Toulouse, France, 2016.

[133] Mircea Negrean. *Performance Analysis of Multi-Core Multi-Mode Systems with Shared Resources - Principles and Application to AUTOSAR -*. Ph.d. thesis, Technische Universität Braunschweig, Institute of Computer and Network Engineering, 2016.

[134] Moritz Neukirchner. *Establishing Sufficient Temporal Independence Efficiently - A Monitoring Approach*. Ph.d. thesis, Technische Universität Braunschweig, Institute of Computer and Network Engineering, 2014.

[135] NXP Semiconductors N.V. SJA1105 - 5-port automotive Ethernet switch. http://www.nxp.com/documents/data_sheet/SJA1105.pdf, 2016.

[136] NXP Semiconductors N.V. TJA1100 100BASE-T1 PHY for Automotive Ethernet Data Sheet, 2017.

[137] OPEN Alliance SIG. http://www.opensig.org/home/. Accessed: 2015-05-30.

[138] OPEN Alliance SIG. Requirements for Ethernet Switch Semiconductors. http://www.opensig.org/download/document/197/TC11_Ethernet_Switch_Requirements_v1_0.pdf, 2016.

[139] Kostas Pagiamtzis and Ali Sheikholeslami. Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey. *IEEE Journal of Solid-State Circuits*, 41(3):712–727, March 2006.

[140] Paul Pop, Michael Lander Raagaard, Silviu S. Craciunas, and Wilfried Steiner. Design optimisation of cyber-physical distributed systems using ieee time-sensitive networks. *IET Cyber-Physical Systems: Theory & Applications*, 1(1):86–94, 2016.

[141] Jon Postel. User Datagram Protocol. RFC 768, RFC Editor, August 1980.

[142] R. Bonica and F. Baker and G. Huston and R. Hinden and O. Troan and F. Gont. IP Fragmentation Considered Fragile. RFC 8900, RFC Editor, September 2020.

[143] R. Hofmann and B. Nikolic and R. Ernst. Challenges and limitations of ieee 802.1cb-2017. *IEEE Embedded Systems Letters*, pages 1–1, 2019.

[144] R. Vida and L. Costa. Internet Group Management Protocol, Version 3. RFC 3810, RFC Editor, June 2004.

[145] F. Reimann, S. Graf, F. Streit, M. Glas, and J. Teich. Timing analysis of Ethernet AVB-based automotive E/E architectures. In *IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, 2013.

[146] Niklas Reusch, Luxi Zhao, Silviu S. Craciunas, and Paul Pop. Window-based schedule synthesis for industrial ieee 802.1qbv tsn networks. In *2020 16th IEEE International Conference on Factory Communication Systems (WFCS)*, pages 1–4, 2020.

[147] Kasper Revsbech, Henrik Schiøler, Tatiana K. Madsen, and Jimmy J. Nielsen. Worst-case Traversal Time Modelling of Ethernet Based In-Car Networks Using Real Time Calculus. In *Proceedings of NEW2AN*. Springer-Verlag, 2011.

[148] Kai Richer. *Compositional Scheduling Analysis Using Standard Event Models*. Ph.d. thesis, Technische Universität Braunschweig, Institute of Computer and Network Engineering, 2004.

[149] Robert Bosch GmbH. *CAN Specification Version 2.0*. 1991.

[150] Jonas Rox and Rolf Ernst. Formal Timing Analysis of Full Duplex Switched Based Ethernet Network Architectures. In *SAE World Congress*, volume System Level Architecture Design Tools and Methods (AE318), 2010.

[151] S. Quinton and M. Hanke and R. Ernst. Formal analysis of sporadic overload in real-time systems. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 515–520, March 2012.

[152] SAE International. J3016 - Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems, January 2014.

[153] Simon Schliecker. *Performance Analysis of Multiprocessor Real-Time Systems with Shared Resources*. Ph.d. thesis, Technische Universität Braunschweig, Institute of Computer and Network Engineering, 2011.

[154] Jens Schmitt and Frank Zdarsky. The disco network calculator: a toolbox for worst case analysis. page 8, 01 2006.

[155] Jens B. Schmitt, Frank A. Zdarsky, and Ivan Martinovic. Improving performance bounds in feed-forward networks by paying multiplexing only once. In *14th GI/ITG Conference - Measurement, Modelling and Evalutation of Computer and Communication Systems*, pages 1–15, 2008.

[156] R. Schneider, L. Zhang, D. Goswami, A. Masrur, and S. Chakraborty. Compositional analysis of switched Ethernet topologies. In *Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1099–1104, March 2013.

[157] Ramon Serna Oliver, Silviu S. Craciunas, and Wilfried Steiner. Ieee 802.1qbv gate control list synthesis using array theory encoding. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–24, 2018.

[158] M. Shreedhar and George Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '95. ACM, 1995.

[159] J. Specht and S. Samii. Urgency-Based Scheduler for Time-Sensitive Switched Ethernet Networks. In *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 75–85, July 2016.

[160] Statistisches Bundesamt (Destatis). Verkehrsunfälle 2018. July 2019.

[161] Steffen Stein. *Allowing Flexibility in Critical Systems: The EPOC Framework*. Ph.d. thesis, Technische Universität Braunschweig, Institute of Computer and Network Engineering, 2012.

[162] Wilfried Steiner. An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks. In *2010 31st IEEE Real-Time Systems Symposium*, pages 375–384, 2010.

[163] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. RFC 2001, RFC Editor, January 1997.

[164] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2002.

[165] Michael Johas Teener. Back to the Future: Using TAS and Preemption for Deterministic Distributed Delays. In *IEEE 802.1 AVB TG Meeting*, San Antonio, TX, USA, November 2012.

[166] Sivakumar Thangamuthu, Nicola Concer, Pieter J. L. Cuijpers, and Johan J. Lukkien. Analysis of Ethernet-switch Traffic Shapers for In-vehicle Networking Applications. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[167] Daniel Thiele, Philip Axer, Rolf Ernst, and Jan R. Seyler. Improving Formal Timing Analysis of Switched Ethernet by Exploiting Traffic Stream Correlations. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, New Delhi, India, October 2014.

[168] Daniel Thiele and Rolf Ernst. Formal Analysis Based Evaluation of Software Defined Networking for Time-Sensitive Ethernet. In *Proc. of Design, Automation, and Test in Europe (DATE)*, Dresden, Germany, March 2016.

[169] Daniel Thiele and Rolf Ernst. Formal worst-case performance analysis of time-sensitive Ethernet with frame preemption. In *International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–9, Sep. 2016.

[170] Daniel Thiele and Rolf Ernst. Formal Worst-Case Timing Analysis of Ethernet TSN's Burst-Limiting Shaper. In *Proc. of Design, Automation, and Test in Europe (DATE)*, Dresden, Germany, March 2016.

[171] Daniel Thiele, Rolf Ernst, and Jonas Diemer. Formal Worst-Case Timing Analysis of Ethernet TSN's Time-Aware and Peristaltic Shapers. In *IEEE Vehicular Networking Conference (VNC)*, December 2015.

[172] Daniel Thiele, Johannes Schlatow, Philip Axer, and Rolf Ernst. Formal timing analysis of CAN-to-Ethernet gateway strategies in automotive networks. *Real-Time Systems*, 2015.

[173] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *The 27th Annual International Symposium on Computer Architecture (ISCA)*, volume 4, 2000.

[174] Ludovic Thomas and Jean-Yves Le Boudec. On time synchronization issues in time-sensitive networks with regulators and nonideal clocks. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 4(2), jun 2020.

[175] Ludovic Thomas, Jean-Yves Le Boudec, and Ahlem Mifdaoui. On cyclic dependencies and regulators in time-sensitive networks. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 299–311, 2019.

[176] Ludovic Thomas, Ahlem Mifdaoui, and Jean-Yves Le Boudec. Worst-case delay bounds in time-sensitive networks with packet replication and elimination. *CoRR*, abs/2110.05808, 2021.

[177] K. W. Tindell, A. Burns, and A. J. Wellings. An extensible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems Journal*, 6:133–151, 1994.

[178] M. Traub, A. Maier, and K. L. Barbehön. Future automotive architecture and the impact of it trends. *IEEE Software*, 34(3), May 2017.

[179] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, and Per Stenström. The worst-case execution-time problem - overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, 7(3), May 2008.

[180] Ludwig Winkel and Michael Johas Teener. Real-time Ethernet on IEEE 802.3 Networks. In *IEEE 802.3 Plenary Meeting*, Berlin, Germany, 2015.

[181] Jinli Yan, Wei Quan, Xuyan Jiang, and Zhigang Sun. Injection time planning: Making cqf practical in time-sensitive networking. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 616–625, 2020.

[182] Luxi Zhao, Paul Pop, Zijie Gong, and Bingwu Fang. Improving latency analysis for flexible window-based gcl scheduling in tsn networks by integration of consecutive nodes offsets. *IEEE Internet of Things Journal*, 8(7):5574–5584, 2021.

[183] Luxi Zhao, Paul Pop, and Sebastian Steinhorst. Quantitative performance comparison of various traffic shapers in time-sensitive networking. *ArXiv*, abs/2103.13424, 2021.

[184] Luxi Zhao, Paul Pop, Zhong Zheng, Hugo Daigmorte, and Marc Boyer. Latency analysis of multiple classes of avb traffic in tsn with standard credit behavior using network calculus. *IEEE Transactions on Industrial Electronics*, 68(10):10291–10302, 2021.

[185] Michael Ziehensack. Automotive Ethernet for Virtual Machines. In *IEEE-SA Ethernet & IP @ Automotive Technology Day*, London, March 2018.

[186] Helge Zinner, Julian Brand, and Daniel Hopf. Automotive E/E Architecture evolution and the impact on the network. In *IEEE802 Plenary*, March 2019.