Mälardalen University Press Dissertations
No. 402

CONFIGURATION AND TIMING ANALYSIS OF TSN-BASED DISTRIBUTED EMBEDDED SYSTEMS

Bahar Houtan

2024

Mälardalen University

School of Innovation, Design and Engineering

ISBN 978-91-7485-634-7
ISSN 1651-4238
Printed by E-Print AB, Stockholm, Sweden

Mälardalen University Press Dissertations
No. 402

CONFIGURATION AND TIMING ANALYSIS OF TSN-BASED DISTRIBUTED EMBEDDED SYSTEMS

Bahar Houtan

Akademisk avhandling

som för avläggande av teknologie doktorsexamen i datavetenskap vid Akademin för innovation, design och teknik kommer att offentligen försvaras tisdagen den 12 mars 2024, 09.15 i Gamma och via Zoom, Mälardalens universitet, Västerås.

Fakultetsopponent: Professor Jean-Luc Scharbarg, University of Toulouse (France)

Mälardalen University

Akademin för innovation, design och teknik

Abstract

The set of IEEE Time-Sensitive Networking (TSN) standards is an emerging candidate for backbone communication in modern applications of real-time distributed embedded systems. TSN provides various traffic shaping mechanisms that aim at managing the timing requirements of traffic. Emerging applications of these systems, particularly in the automotive domain, often run complex distributed software that requires low-latency and high-bandwidth communication across multiple onboard electronic control units. Using TSN in these systems introduces multiple challenges. Specifically, the developers of these systems face a lack of development techniques and tools, as TSN standards only offer general recommendations for the use of its features and mechanisms. There is an urgent need for development techniques, tools, and methods to assist the developers in effectively leveraging the features outlined in TSN standards. In this thesis, we identify and address several challenges encountered in the development of TSN-based distributed embedded systems, particularly focusing on the stages of system and software modeling, network configuration, and timing analysis. The overall goal of this thesis is to support the development of these systems in the aforementioned stages while considering the Quality of Service (QoS) requirements of all traffic classes in TSN. We present techniques to facilitate the system and software modeling of TSN-based distributed embedded systems. These techniques enable performing timing analysis in the early stages of system and software development. In the stage of network configuration, we propose techniques for managing the configuration complexity and supporting the automatic configuration of mechanisms in TSN. The proposed configuration techniques consider achieving acceptable QoS in various traffic classes. In the stage of timing analysis, we address the challenges of incorporating various TSN traffic classes and mechanisms by extending the existing timing analyses. The results indicate that the proposed techniques effectively facilitate the system and software modeling, network configuration, and timing analysis of TSN-based distributed embedded systems.

ISBN 978-91-7485-634-7
ISSN 1651-4238

”Peace comes through bearing the hardships. As the treasure and the dragon, the rose and the thorn, joy and sorrow, all mingle into one.”

Saa’di Shiraazi

Abstract

The set of IEEE Time-Sensitive Networking (TSN) standards is an emerging candidate for backbone communication in modern applications of real-time distributed embedded systems. TSN provides various traffic shaping mechanisms that aim at managing the timing requirements of traffic. Emerging applications of these systems, particularly in the automotive domain, often run complex distributed software that requires low-latency and high-bandwidth communication across multiple onboard electronic control units. Using TSN in these systems introduces multiple challenges. Specifically, the developers of these systems face a lack of development techniques and tools, as TSN standards only offer general recommendations for the use of its features and mechanisms. There is an urgent need for development techniques, tools, and methods to assist the developers in effectively leveraging the features outlined in TSN standards. In this thesis, we identify and address several challenges encountered in the development of TSN-based distributed embedded systems, particularly focusing on the stages of system and software modeling, network configuration, and timing analysis. The overall goal of this thesis is to support the development of these systems in the aforementioned stages while considering the Quality of Service (QoS) requirements of all traffic classes in TSN. We present techniques to facilitate the system and software modeling of TSN-based distributed embedded systems. These techniques enable performing timing analysis in the early stages of system and software development. In the stage of network configuration, we propose techniques for managing the configuration complexity and supporting the automatic configuration of mechanisms in TSN. The proposed configuration techniques consider achieving acceptable QoS in various traffic classes. In the stage of timing analysis, we address the challenges of incorporating various TSN traffic classes and mechanisms by extending the existing timing analyses. The results indicate that the proposed techniques effectively facilitate the system and software modeling, network configuration, and timing analysis of TSN-based distributed embedded systems.

Sammanfattning

IEEE Time-Sensitive Networking (TSN) är en framväxande kommunikationsteknik för moderna tillämpningar inom distribuerade inbyggda realtidssystem. TSN definerar olika mekanismer för trafikomformning som syftar till att ge flexibilitet i hur fördröjningar och bandbredd fördelas mellan olika användare. Nya tillämpningar av distribuerade inbyggda system i realtid har ofta mycket komplex distribuerad programvara som kräver kommunikation med låg latens och hög bandbredd över flera enheter (ECU i automotiv system). Att anpassa sig till TSN för att utveckla sådana system innebär många utmaningar. Dessutom finns det en brist på stödtekniker för utvecklare eftersom TSN-standarder endast ger generaliserade rekommendationer för tillämpningen av dess mekanismer. Därför finns det ett behov av tekniker och metoder för att stödja utvecklare att använda tillgängliga funktioner i TSN-standarderna. I den här avhandlingen angriper vi några av utmaningarna i utvecklingsstadierna av TSN-baserade distribuerade inbyggda system som är modellering, konfiguration av nätverk och tidsanalys. Målet med avhandlingen är att stödja utvecklingen av TSN-baserade distribuerade inbyggda system i dessastadier, samtidigt som man tar hänsyn till Quality of Service (QoS) krav inom alla trafikklasser i TSN. Vi tillhandahåller tekniker för att underlätta system- och mjukvarumodellering av TSN-baserade distribuerade inbyggda system som ger ett verktyg för utvecklare att designa sina system i tidiga stadier av systemutveckling. Inom konfiguration av nätverk föreslår vi tekniker för att hantera komplexiteten och den automatiska konfigurationen av TSN-formningsmekanismer. Under tiden tar vi hänsyn till att nå en acceptabel QoS i olika trafikklasser. Inom tidsanalysen tar vi itu med utmaningarna med att införliva olika TSN-trafikklasser och mekanismer genom att utöka de befintliga tidsanalysmetoderna. Genom de tre bidragna tillvägagångssätten underlättar våra resultat processen att utveckla TSN-baserade distribuerade inbyggda system.

Acknowledgements

I express my gratitude to my supervisors Saad Mubeen, Mohammad Ashjaei, Masoud Daneshtalab, and Mikael Sjödin. All of you played a very important role in my growth and preparation of this thesis. I will always be grateful for that. Thank you Saad for patiently providing me with your insights and comments. I am grateful for the times that you encouraged me to keep up when even I was doubting myself. I would like to truly thank Mohammad, one of my co-supervisors who I see as my second main supervisor. Thank you for always being available for discussions. Either if I came for discussion when it was early in the morning as you arrived at your office, or if I just approached you for discussion in the corridor or via chat you were always open for a talk. Thank you Masoud and Mikael for your valuable advice, suggestions, and encouragement.

Special thanks to John Lundbäck and Kurt-Lennart Lundbäck for allowing me to experience your nice work environment at Arcticus Systems, applying my research in RUBUS ICE, and for the enlightening lunch talks. These had a great impact on shaping my research. Thanks to Sara Afshar, Mehmet Onur Aybek, and Albert Bergström, my co-authors in the included papers in this thesis.

I find myself more than lucky to have met several amazing, creative, motivating, and supportive people in my life and education. I want to express my utmost respect to Professors Mats Björkman, Elizabeth Uhlemann, Thomas Nolte, Cristina Seceleanu, and Jan Carlson. Besides, I extend my thanks and appreciation to all the teachers, researchers, staff, and PhD students that I met and had the opportunity to work with during my PhD studies at Mälardalen University. Listing all of you here is impossible but I want you to know that each of you has a place in my heart.

Finally, I am eternally grateful to my family, especially my parents and brother for their endless support and love. Thank you for giving me the motivation to work hard, to build myself, and to never give up. I owe everything I have achieved in my life to you.

Bahar Houtan,
Västerås, February 2024

This work is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the DESTINE project (grant nr. 2018-02728). We would like to thank our industrial partners, which include Volvo Construction Equipment, Sweden, and Arcticus Systems, Sweden, for their invaluable feedback.

List of Publications

Main Contributing Publications[1]

- **Paper A:** *"An Automated Configuration Framework for TSN Networks,"* B. Houtan, A. Bergström, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, in the 22^{nd} IEEE International Conference on Industrial Technology, Valencia, Spain, 2021.
- **Paper B:** *"Synthesising Schedules to Improve QoS of Best-Effort Traffic in TSN Networks,"* B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, in the Proceedings of the 29th International Conference on Real-Time Networks and Systems, ACM, Nantes, France, 2021.
- **Paper C:** *"Schedulability Analysis of Best-Effort Traffic in TSN Networks,"* B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Afshar, and S. Mubeen, in the 26^{th} IEEE International Conference on Emerging Technologies and Factory Automation, Västerås, Sweden, 2021.
- **Paper D:** *"Supporting End-to-End Data Propagation Delay Analysis for TSN-Based Distributed Vehicular Embedded Systems,"* B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, in the Journal of Systems Architecture, vol. 141, 2023.
- **Paper E:** *"End-to-end Timing Modeling and Analysis of TSN in Component-Based Vehicular Software,"* B. Houtan, M. O. Aybek, M. Ashjaei, M. Daneshtalab, M. Sjödin, J. Lundbäck, and S. Mubeen, in the 26^{th} IEEE International Symposium on Real-Time Distributed Computing, Nashville, Tennessee, USA, 2023.
- **Paper F:** *"Bandwidth Reservation Analysis for Schedulability of AVB Traffic in TSN,"* B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, in the 25^{th} IEEE International Conference on Industrial Technology, Bristol, UK, 2024.

[1]These publications are included in the Ph.D. Thesis.

Other Related Publications[2]

- *End-to-end Timing Model Extraction from TSN-Aware Distributed Vehicle Software*, B. Houtan, M. O. Aybek, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Mubeen, in the 48th Euromicro Conference Series on Software Engineering and Advanced Applications, 2022.
- *Configuring and Analysing TSN Networks Considering Low-Priority Traffic*, B. Houtan, Licentiate Thesis, Mälardalen University, Västerås, Sweden, 2021.
- *Supporting End-to-end Data-propagation Delay Analysis for TSN Networks*, B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Mubeen, MRTC Report, Mälardalen Real-Time Research Centre, 2021, Report Nr. MDH-MRTC-339/2021-1-SE.
- *Developing Predictable Vehicular Embedded Systems Utilizing Time-Sensitive Networking–A Research Plan*, B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Mubeen, in the 15th Swedish National Computer Networking Workshop, 2019.
- *Work in Progress: Investigating the Effects of High Priority Traffic on the Best Effort Traffic in TSN Networks*, B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Mubeen, in the 40th IEEE Real-Time Systems Symposium, 2019.

[2]These publications are not included in the Ph.D. thesis.

Acronyms

AADL	Architecture Analysis & Design Language.
AUTOSAR	AUTomotive Open System ARchitecture.
AVB	Audio-Video Bridging.
BE	Best-Effort.
BET	Bounded Execution Time.
BRA	Bandwidth Reservation Analysis.
CAN	Controller Area Network.
CAN FD	CAN Flexible Data Rate.
CBS	Credit-Based Shaping.
CP	Constraint Programming.
ECUs	Electronic Control Units.
FIFO	First-In-First-Out.
GA	Genetic Algorithm.
GCL	Gate Control List.
GUI	Graphical User Interface.
ICE	Integrated Component model development Environment.
ILP	Integer Linear Programming.
LET	Logical Execution Time.

ML	Machine Learning.
NC	Network Calculus.
OMT	Optimization Modulo Theorem.
OPC UA	OPC Unified Architecture.
QoC	Quality of Control.
QoS	Quality of Service.
RC	Rate-Constrained.
RCM	RUBUS Component Model.
RG	Research Goal.
RGs	Research Goals.
ROS	Robot Operating System.
RTA	Response-Time Analysis.
SA	Simulated Annealing.
SMT	Satisfiability Modulo Theorem.
SP	Strict Priority.
ST	Scheduled Traffic.
SWC	Software Components.
TAS	Time-Aware Shaper.
TC	Thesis Contribution.
TCs	Thesis Contributions.
TSN	Time-Sensitive Networking.
TT	Time-Triggered.
TTEthernet	Time-Triggered Ethernet.
WCET	Worst Case-Execution Time.

Contents

Part I

Thesis

Chapter 1

Introduction

Several standardized protocols are used to support high-bandwidth communication requirements among the Electronic Control Units (ECUs) in the vehicular domain[1]. Examples of some commonly used onboard communication protocols in the automotive domain include Controller Area Network (CAN) [1], CAN Flexible Data Rate (CAN FD) [2], and Flex Ray [3]. The set of Time-Sensitive Networking (TSN) [4] standards is a relatively recent candidate that further support real-time onboard communication. The TSN standards introduce various mechanisms [5], such as time-synchronization (IEEE802.1AS), bounded latency (IEEE802.1Qav, IEEE 802.1Qbu, and IEEE 802.1Qbv), reliability (IEEE 802.1CB), and security (IEEE802.1Qci). With the support for traffic classes with different timing requirements, i.e., hard real-time Scheduled Traffic (ST), high bandwidth Audio-Video Bridging (AVB) traffic, and Best-Effort (BE) traffic, TSN serves for the flexible design of highly complex functions for distributed embedded systems. However, guaranteeing the timing predictability of the time-critical functions, while supporting an acceptable level of Quality of Service (QoS) for non-critical functions is a challenge in the design and development of TSN-based automotive systems [6, 7].

[1]Vehicular domain includes systems like cars, trucks, construction vehicles, loading vehicles, recycling vehicles, and moving cranes, to mention a few. The automotive domain refers to a subset of the vehicular domain.

1.1 Research Goals

Towards facilitating the development of distributed embedded systems that use TSN as the real-time communication backbone, we have formulated the overall research goal of this thesis as follows:

To support the development of TSN-based distributed embedded systems in system and software modeling, network configuration, and timing analysis stages while considering the QoS requirements of all traffic classes in TSN.

The overall research goal targets the improvement of the process for developing TSN-based distributed embedded systems. For this, we consider three main development stages of these systems: system and software modeling, network configuration, and timing analysis. We aim at supporting the three development stages to facilitate reaching a satisfactory or improved QoS for all traffic classes depending on the application requirements. The overall research goal is divided into three Research Goals (RGs) as we explain in the following subsections.

1.1.1 Research Goal 1 ($\mathbf{RG_1}$)

The focus of RG_1 is on the timing analysis stage in developing TSN-based systems and is formulated as follows:

RG_1: Develop an end-to-end data-propagation delay analysis for TSN-based distributed embedded systems.

TSN-based distributed embedded systems comprise multiple end-stations communicating over TSN. Analyzing the end-to-end data-propagation delays in these systems requires pre-calculated response times of tasks in the end-stations and response times of TSN messages as essential input parameters. Calculations of these delays is challenging for several reasons. Firstly, the existing schedulability analysis techniques do not support all the configuration scenarios of TSN mechanisms. For example, using new combination of TSN mechanisms, i.e, frame preemption, hold and release, CBS, and TAS leads to implications for the design and analysis of these systems [8], whereas the existing worst-case Response-Time Analysis (RTA) supports only the most common design scenarios. For instance, there is a lack of worst-case RTA to verify timing requirements of real-time traffic if it is assigned to the lower-priority BE class, i.e., legacy real-time traffic. Secondly, TSN supports synchronization of the sender and receiver end-stations for deterministic transmission of real-time ST according to IEEE 802.1AS. The existing end-to-end data-propagation delay analysis supports only non-synchronized legacy communication protocols, such as CAN [1]. The existing analysis calculates pessimistic end-to-end delays (data age and reaction time delays) in the case when the sender and receiver end-stations are synchronized. Hence, the existing analysis needs extensions to support distributed embedded systems where the end-stations are synchronized.

In summary, the purpose of RG_1 is to address the following gaps: 1) developing RTA for all traffic classes in TSN, and 2) incorporating the newly developed RTA and the TSN synchronization mechanism to the end-to-end data-propagation delay analysis.

1.1.2 Research Goal 2 (RG_2)

The RG_2 addresses the configuration stage in developing TSN-based systems and is formulated as follows:

> *RG_2: Develop network configuration techniques considering the QoS requirements of all traffic classes in TSN-based distributed embedded systems.*

One of the challenges that the designers of the systems face at the network configuration stage is that increasing the complexity of applications also increases the configuration search space for TSN-based distributed embedded systems. Firstly, the complexity lies in the increasing size of the network and traffic. Secondly, the integration of more safety-critical applications into the system functionalities demands utilizing a combination of traffic-shaping mechanisms to manage different timing requirements of traffic. This, in turn, contributes to the configuration complexity because of the need for carefully setting up the traffic shaping mechanisms considering the QoS requirements of the system, i.e., meeting the timing constraints on the data age and reaction time delays.

Traditionally, the configuration of the traffic shapers in the network is conducted based on experience and understanding of the system. However, finding a proper configuration for TSN mechanisms for larger applications can become difficult even for skilled designers. Moreover, the recommendations by the TSN standards are generalized. There is a lack of methods to configure various mechanisms and features in TSN for the scenarios that are not anticipated by the TSN recommendations.

To reach RG_2, we identify that there is a lack of techniques and methods for 1) automatic network configuration; 2) managing the configuration complexity of TSN shaper mechanisms; and 3) reaching an acceptable QoS while using various TSN shaper mechanisms.

1.1.3 Research Goal 3 (RG_3)

The RG_3 targets the system and software modeling stage in developing TSN-based systems and is formulated as follows:

> *RG_3: Develop a methodology to extract end-to-end timing information from TSN-based distributed software architectures to support their timing analysis.*

To perform end-to-end data-propagation delay analysis of a distributed embedded system, it is crucial to extract the necessary timing information from the system. One of the challenges during the development of TSN-based distributed embedded systems is to support their analysis at the software architecture abstraction level. The software architecture of a system can be described in terms of software components and their interconnections. Software architectures often do not completely describe the end-to-end timing information. Some of the timing information needs to be derived via/from automatic configuration tools as a consequence of RG_2. For example, some timing information may come from the network configuration mechanisms. More specifically, the offsets for ST messages can be manually defined and included in the software architecture. However, configuring offsets within large and complex systems can be a significant workload for the software architecture developers/modelers. Therefore, the offset-related information can come from automatic configuration algorithms that are typically preferred for scheduling the ST messages in TSN.

To reach RG_3, we aim at facilitating the system and software modeling stage by addressing the challenge of end-to-end timing analysis of software architectures describing the TSN-based distributed embedded systems.

1.2 Research Process

An overview of the research process of this thesis is included in Figure 1.1. We use the hypothetico-deductive [9] research method for computer science to conduct research towards the overall research goal of this thesis. This research is instantiated in close collaboration with industrial partners: Volvo CE[2] and Arcticus Systems[3]. Therefore, in the process of this research, we received the industrial partners' feedback through discussions.

- **Beginning**: At the beginning of the doctoral studies the overall research goal is defined as a research proposal. This overall research goal is then refined through

[2]https://www.volvoce.com/
[3]https://www.arcticus-systems.com/

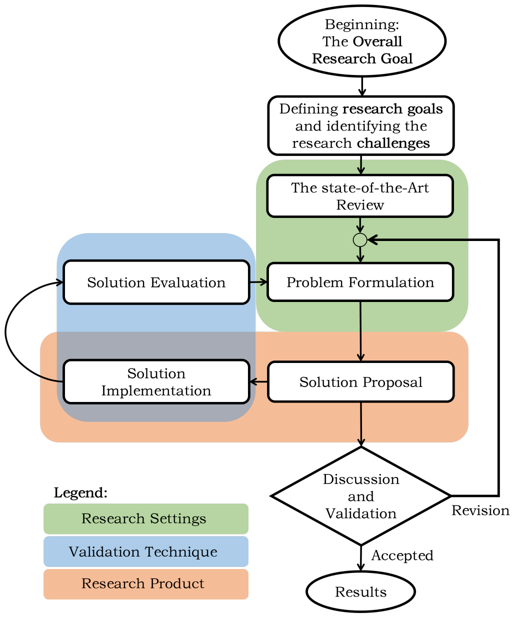


Figure 1.1. Research process

discussions. Consequently, we define the RGs of the thesis. Subsequently, the research challenges that we would face to fulfill RGs are identified through discussions.

- **The state-of-the-art Review**: To obtain insights into the state of the art and also identify the requirements in practice, a literature review is performed towards each of the RGs. Besides, the latest revisions of the set of TSN standards and AUTOSAR standards are referred to iteratively.

- **Problem Formulation**: In this stage, we identify and formulate problems in the domain of each RG. The identified problems address challenges towards each of the RGs. Moreover, through the meetings with our industrial partners at Volvo CE and Arcticus Systems, we develop applicable use-cases of TSN standards towards each RG.

- **Solution Proposal**: Novel solutions that address challenges towards each RG are discussed and proposed. The research methodology and implementation challenges for the solutions are identified. The proposed solutions are discussed with the partners and revised accordingly before proceeding to the implementation step.

- **Solution Implementation** The selected solutions from the previous stage are implemented by applicable techniques and methods, e.g., simulation tools (i.e. OMNET++ and NeSTiNg), satisfiability solvers (i.e. Z3 Solver), theoretical analysis methods (i.e. worst-case RTA and end-to-end data-propagation delay analysis).

- **Solution Evaluation**: Each solution is evaluated on an appropriate use case or using experiments. We select some of the most common use cases and data sets in the automotive domain. Besides, we receive use cases from our industrial partners.

- **Discussion and Validation** The evaluation results are discussed and their validity is examined. If the evaluation results are judged to require a revision, we trace back to identify the source and propose a new solution to the problem. Furthermore, in such cases, the research process is reiterated by refining the current solution or defining a new solution. Finally, if the evaluation results is accepted, the process is finished by publishing each of the proposed solutions.

1.3 Thesis Outline

This thesis consists of two main parts. The first part provides an introduction to the overall work of the thesis and consists of five chapters:

Chapter 1 - Introduction: Chapter 1 presented an introduction to the research area. Besides, this chapter provided a detailed overview of the research in this thesis. We also presented the overall research goal, RGs, and the challenges faced in reaching the RGs. Moreover, the research process used in the thesis was presented.

Chapter 2 - Background and Related Works: In this chapter, we provide technical background on the research area to which this thesis contributes: real-time communication, offline scheduling, schedulability analysis, end-to-end data-propagation delay analysis, and software architecture modeling. Moreover, we present state-of-the-art research and position our contributions within this research landscape.

Chapter 3 - Research Contributions: In this chapter, firstly we present our findings with Thesis Contributions (TCs). Secondly, we provide a mapping of how the TCs contribute to each of the RGs. Thirdly, this chapter provides a summary of the papers included in this thesis.

Chapter 4 - Discussions: This chapter presents a discussion of how the research goals are addressed in this thesis by TCs.

Chapter 5 - Conclusions and Future Works: This chapter concludes the thesis by summarizing our findings and suggesting potential future works related to the work presented in this thesis.

Finally, we present all the publications that are included in this thesis. A collection of six research papers constitutes the second part of this thesis.

Chapter 2

Background and Related Works

This section provides a review of the technical background required to understand the contributions in this thesis. Additionally, we present the state-of-the-art relevant to the research presented in the thesis.

2.1 Real-Time Communication

2.1.1 Switched Ethernet

Switched Ethernet [10] is a protocol used in common switches. It features a maximum of eight First-In-First-Out (FIFO) queues, where each of the queues is associated with a priority. A network message is allocated to a FIFO queue firstly according to its priority. Secondly, the message goes through the queue based on its arrival time in the queue. Strict Priority (SP) algorithm (also known as transmission selection algorithm) allocates data frame to the FIFO queues. The lack of control over the flow of data in the FIFO queues is one of the main limitations of switched Ethernet for real-time communication. For example, there is no mechanism in switched Ethernet to prevent blocking by lower-priority messages. Moreover, the switched Ethernet hardware architecture allows only a limited number of (maximum eight queues) which limits the efficiency of its priority-based scheduling.

2.1.2 Time-Triggered Ethernet (TTEthernet)

TTEthernet [11] protocol enhances the conventional Ethernet to support three traffic classes namely, Time-Triggered (TT), Rate-Constrained (RC), and Best-Effort (BE). The

TT class allows offline scheduling of Ethernet frames to control the transmission flow of data. In particular, TTEthernet applies to traffic with strict real-time requirements in a network with globally synchronized switches and end-stations. Traffic with less strict deadlines are set as RC. The network can be configured to efficiently let RC traffic utilize a portion of the remaining bandwidth from TT schedules. Therefore, we can only guarantee that the delay of RC class is within a certain bound. It is worth noting that TTEthernet is also capable of functioning similarly to conventional switched Ethernet for the BE class.

2.1.3 Time-Sensitive Networking (TSN)

The IEEE Time-Sensitive Networking (TSN) standards [4] fill the gaps of the previous generations of Ethernet with features applicable to distributed real-time embedded systems. TSN enhances various traffic shaping mechanisms and includes several classes of traffic, i.e., ST, AVB, and BE (Section 8.6.8 of TSN standards [5]). The traffic shaping mechanisms are designed based on the timing requirements of TSN traffic classes. In this thesis, we particularly focus on clock synchronization, Time-Aware Shaper (TAS), and Credit-Based Shaping (CBS) mechanisms in TSN.

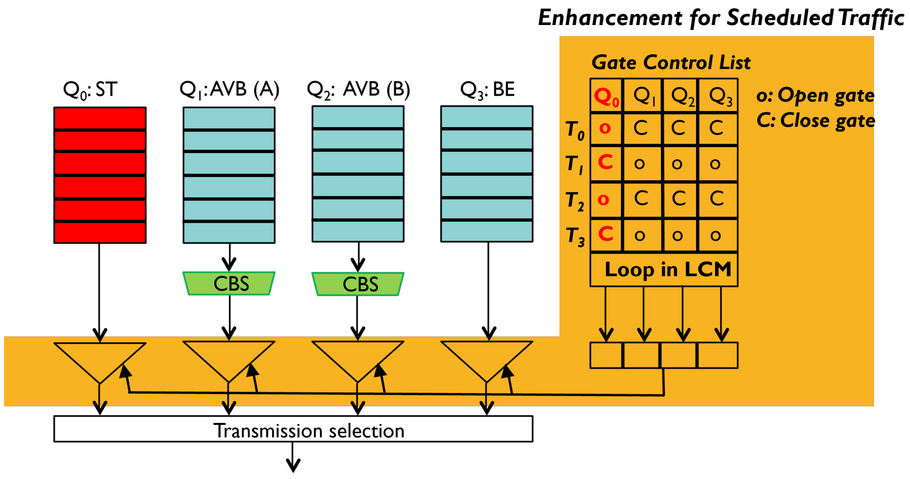


Figure 2.1. Time-Aware Shaper (TAS).

Clock Synchronization: A set of methods that provide a reliable timing reference for the correct time-stamping of events, the right sequencing of operations, and deterministic message delivery in real-time networks is known as clock synchronization. The TSN standards [5] support synchronization of end-stations according to the IEEE 802.1AS standard. In particular, the end-stations should be synchronized when the ST class is used, according to the IEEE 802.1Qbv standard. But in general, when the IEEE 802.1AS standard is used then the end-stations in the TSN network should be considered synchronized regardless of which TSN traffic class, ST, AVB or BE, is used.

Time-Aware Shaper (TAS): TAS mechanism provides dedicated queues for real-time traffic as defined in the IEEE 802.1Qbv standard for deterministic transmission of ST. TAS as depicted in Figure 2.1 uses offline schedules which are time-stamped gate states defined in the Gate Control List (GCL). Unlike TTEthernet in which we schedule frames, in the case of TSN we can schedule a stream of data in ST class (queues) thanks to the GCL. The rows of GCL indicate the time and order of enabling the gates connected to the queues in the TSN switch. Figure 2.1 shows a simplified example of TSN egress port with four queues, however, there can be a maximum of eight queues that can be controlled by eight-bit rows of GCL. ST queues preempt the transmission of lower-priority traffic. The talker and listener end-stations must meet the synchronization requirements according to IEEE 802.1AS standard so that the offline schedules guarantee the low-jitter and predictable transmission of the frames. In Section 2.2, we explain related works in offline scheduling.

Credit-Based Shaping (CBS): The CBS mechanism reserves bandwidth for the AVB classes while it also limits the AVB traffic to interfere with traffic in the lower-priority classes. CBS can be enabled for all of the eight available queues per port of a TSN switch in the standards [5], thus it can include a maximum of eight classes. It is very common to use only classes A and B in analysis and practice.

The credit of AVB can have the following states:

- **Positive credit**: A message in an AVB class starts its transmission when it is ready for transmission and the credit of the class is greater than or equal to zero. If at the same time, there are other active sources of interference and blocking using the bandwidth, the message must wait. The credit increases while this message waits for available bandwidth.
- **Frozen credit**: An ST message can preempt AVB traffic. During the preemption, the credit of the AVB class freezes until the transmission of the ST message is completed.

- **Negative credit**: the credit decreases while a message in this class uses the bandwidth. If a message is activated while the credit is below zero, it will not be transmitted. The message must wait until the credit is replenished to zero, or goes above zero, i.e., until the bandwidth is free for its transmission. When there are no messages ready for transmission in the class and the credit is negative, the credit increases until it reaches zero.

2.2 Offline Scheduling

According to [12], offline scheduling is a method in which all scheduling decisions are pre-computed before the run time of the system. Therefore, offline scheduling is also known as pre-run-time scheduling. To perform offline scheduling, we require complete knowledge of the task set and its constraints, such as deadlines, computation times, precedence constraints, etc.

ST in TSN must be scheduled offline. Different variations can be included in the ST scheduling problem [13, 14], to name a few, routing of the traffic, preemption, porosity in the schedules, redundant transmissions, or system-level scheduling are some of the parameters to be taken into account. The complexity of the ST scheduling problem depends on the aforementioned variations. On one hand, some variations increase the schedule solution search space. For example, if we combine routing and scheduling of ST. On the other hand, other variations can remove some schedule solutions from the search space, i.e., synthesizing porous schedules.

There are two main approaches to address the ST scheduling problems [15]. Firstly, the exact approaches that use Satisfiability Modulo Theorem (SMT)/ Optimization Modulo Theorem (OMT) to perform an exhaustive search between all the possible offline schedule solutions. This approach is exponentially time-complex in the case of large-scale networks. Secondly, there are more efficient heuristics/meta-heuristic approaches that find a locally optimized offline scheduling solution in a shorter time than the exact approaches.

In this section, we focus on the related works in literature contributing to each of the aforementioned approaches under different offline scheduling problem variations. Figure 2.2 shows the trend in the related works in offline scheduling in TTEthernet and TSN.

2.2.1 Exact Approaches

The seminal work [16] introduces a set of scheduling constraints for TTEthernet in a multi-hop network, which later was used as the foundation of other scheduling solutions

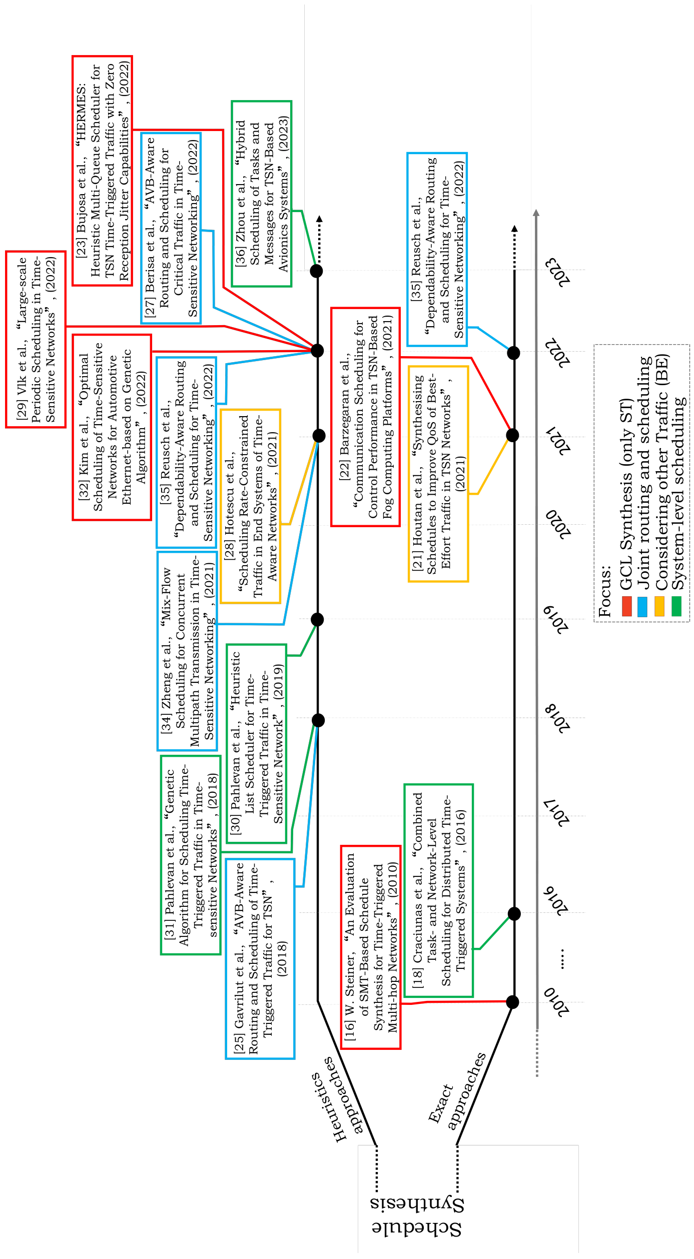

Schedule Synthesis
Heuristics approaches
Exact approaches
2010
...
2016
2017
2018
2019
2020
2021
2022
2023
[16] W. Steiner, "An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks", (2010)
[18] Craciunas et al., "Combined Task- and Network-Level Scheduling for Distributed Time-Triggered Systems", (2016)
[21] Houtan et al., "Synthesising Schedules to Improve QoS of Best-Effort Traffic in TSN Networks", (2021)
[22] Barzegaran et al., "Communication Scheduling for Control Performance in TSN-Based Fog Computing Platforms", (2021)
[35] Reusch et al., "Dependability-Aware Routing and Scheduling for Time-Sensitive Networking", (2022)
[25] Gavrilut et al., "AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN", (2018)
[31] Pahlevan et al., "Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-sensitive Networks", (2018)
[30] Pahlevan et al., "Heuristic List Scheduler for Time-Triggered Traffic in Time-Sensitive Network", (2019)
[34] Zheng et al., "Mix-Flow Scheduling for Concurrent Multipath Transmission in Time-Sensitive Networking", (2021)
[28] Hotescu et al., "Scheduling Rate-Constrained Traffic in End Systems of Time-Aware Networks", (2021)
[35] Reusch et al., "Dependability-Aware Routing and Scheduling for Time-Sensitive Networking", (2022)
[32] Kim et al., "Optimal Scheduling of Time-Sensitive Networks for Automotive Ethernet-based on Genetic Algorithm", (2022)
[29] Vlk et al., "Large-scale Periodic Scheduling in Time-Sensitive Networks", (2022)
[23] Bujosa et al., "HERMES: Heuristic Multi-Queue Scheduler for TSN Time-Triggered Traffic with Zero Reception Jitter Capabilities", (2022)
[27] Berisa et al., "AVB-Aware Routing and Scheduling for Critical Traffic in Time-Sensitive Networking", (2022)
[36] Zhou et al., "Hybrid Scheduling of Tasks and Messages for TSN-Based Avionics Systems", (2023)
Focus:
🟥 GCL Synthesis (only ST)
🟦 Joint routing and scheduling
🟨 Considering other Traffic (BE)
🟩 System-level scheduling

Figure 2.2. Overview of the network offline scheduling approaches.

that used exact approaches. These constraints are as follows.

- **Constraint on the frame size**: The frame constraint ensures that the value of the offset on one link does not force the arrival of the message after its next period of activation.
- **Constraint on the overlapping of messages on a link**: The overlapping constraint checks whether the offset of two different messages on the same link will not cause overlapping time slots for these two messages.
- **Constraint on the order of traversed links**: The order of the offset for the same message on subsequent links must consider the propagation of the message from the source end-station link to the sink end-station link (avoiding time travel).
- **Constraint on meeting the deadline**: The offsets per links in the route of the message must enforce the message to arrive at the destination end-station before the message's deadline (implicitly the next period activation of the sending task).

Later the work in [17], proposed a post-processing heuristic algorithm that receives the TT schedules as input and generates porosities in TT schedules to leave bandwidth that could be efficiently used by RC traffic. Furthermore, the work in [18] combines task-level and network-level offline scheduling for TTEthernet. For this, extra constraints to incorporate the scheduling of Time-Triggered (TT) tasks[1] are defined in [18].

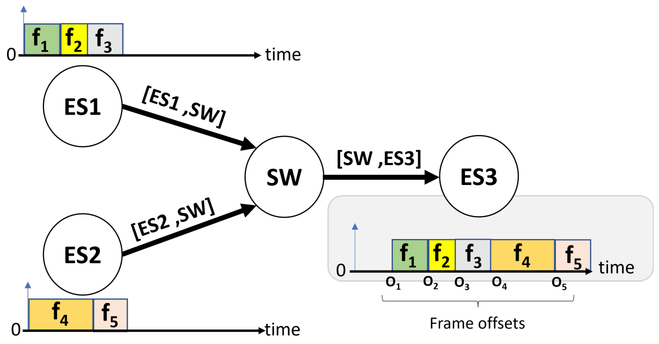
Some of these additional scheduling constraints are as follows:

- **Constraint on the precedence of tasks within end-stations:** This constraint can be included if there exists specific dependencies between the tasks within the chain of tasks inside the talker (sender) or listener (receiver) end-stations; and
- **Network switch memory constraints:** This constraint defines the effect of the buffering approach inside network switches on the traffic. Since, the physical memory in the network switches can use different assumptions for buffering network frames.

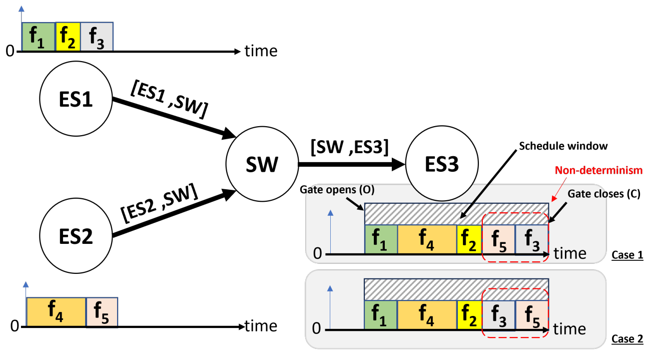
The aforementioned constraints for TTEthernet also apply to TSN, although they are not enough to generate deterministic ST schedules. In TTEthernet the offline schedules are defined for each frame, whereas in TSN schedule windows are defined by scheduling the opening and closing times of the gates. This is because TSN aims at providing flexibility of configuration from a larger perspective, i.e., scheduling streams instead of frames.

[1]Time-Triggered (TT) task is a task that can be independently triggered by an event source, e.g., a periodic clock.

To show the difference between the scheduling problems in TTEthernet and TSN, we present an example in Figure 2.3. In Figure 2.3, $ES1$ and $ES2$ are two end-stations that are sending messages to $ES3$ via the switch SW. We assume that the messages have different sizes. The switch in Figure 2.3(a) is a TTEthernet switch. Since an offset (denoted by O_x) is defined for each of the frames the order of receiving the frames at $ES3$ is exactly known. Whereas, we can create a scheduling window for the same set of traffic in the case of using TSN as shown in Figure 2.3(b). However in this type of schedule, we do not have any control over the flow of frames the order of frames inside the queues can become non-deterministic.



(a) Scheduling frames in TTEthernet.



(b) Gate scheduling in TSN.

Figure 2.3. Comparing offline scheduling in TTEthernet and TSN.

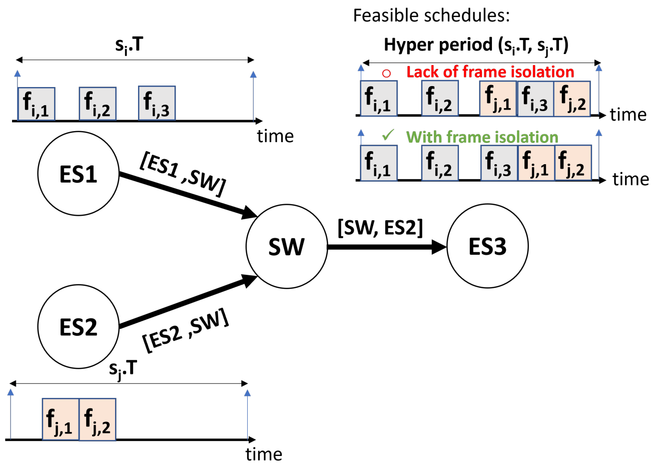
The work in [19] solves the aforementioned non-determinism at the frame level by proposing two constraints, called the frame and stream isolation constraints as shown in the example in Figure 2.4. The isolation constraints ensure the non-interleaving transmission of frames through the same queue (temporal isolation). Moreover in [19], the authors introduce an alternative isolation that restricts the transmission of streams by scheduling them in different queues (spatial isolation).

Figure 2.4(a) is an example of isolating the transmission of every frame of different streams in the queue. This scenario is depicted in Figure 2.4 assuming two streams denoted by s_i and s_j have the same periods (periods of s_i and s_j are subsequently denoted by $s_i.T$ and $s_j.T$). At the right side of Figure 2.4(a) we see the transmission trace on link $[SW, ES3]$ with two possible schedules. The transmission trace on top shows the interleaving of frames of the two streams.

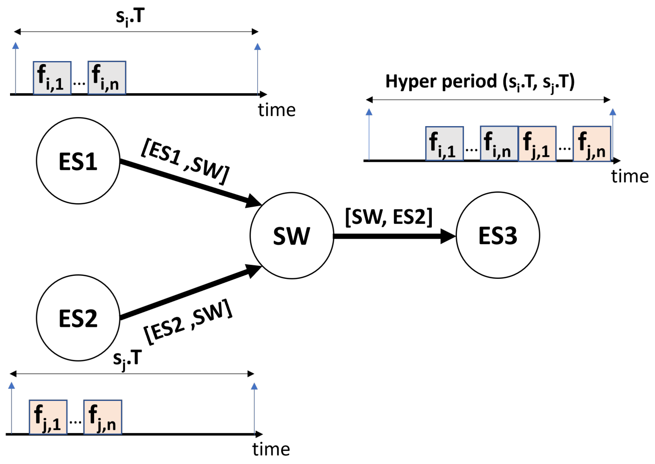
Besides, the transmission of multiple streams via the same queue can cause variations in the arrival times of the frames belonging to different streams for preventing this. Figure 2.4(b) shows how we can isolate the streams. According to the stream isolation constraint when the first frame of a stream is scheduled for transmission (s_j in this example) no other frames of other streams are scheduled until all frames of s_j are dispatched (s_i in this example). In Figure 2.4, for simplicity of illustration, we assumed that the periods of s_i and s_j are the same, hence the hyper period is the same as the periods of the two streams.

Porous scheduling can be chosen for different purposes. For example, The work in [20] creates slacks in the offline schedules of the frames (in TTEthernet). These slacks are used at the run-time of the network e.g. for repairing the routing in the case of unpredictable changes such as link failure. Moreover in [21], we propose a porous offline scheduling solution and present simulation results on the impact of schedule slacks on the end-to-end delays of the BE traffic. The results in [21] we show that scheduling ST streams closer to their deadlines will lead to a better QoS for BE traffic (i.e. reduced response times and fewer deadline misses). An alternative for that is to generate slacks between the ST frames for the benefit of BE frames which is in turn also faster than maximizing ST offsets.

Figure 2.5 shows examples of schedule solutions. In Figure 2.5(a) time stamps $[0, T_0]$, $[T_1, T_2]$ and $[T_3, T_4]$ are reserved for ST frames. The time slots represented by white boxes in Figure 2.5(a) indicate the times when the gates are open for transmission of BE frames. The downward arrows show the deadline of each BE frame. Based on this schedule, ST frames that are packed subsequently within the time slot $[T_1, T_2]$, cause deadline miss for the BE frame, $BE3$. On the other hand, Figure 2.5(b) shows the sparsification of ST frames within time stamp $[T_1, T_2]$ from the previous example into new scheduled time stamps $[T_1, T_2]$, $[T_3, T_4]$ and $[T_5, T_6]$. Consequently, by just adding a few more gate state changes, all BE frames meet their deadlines. The work in [22] considers scheduling ST streams with a perspective on the QoC (control performance). QoC is defined as the variations in the end-to-end delays of the system.



(a) Frame isolation.

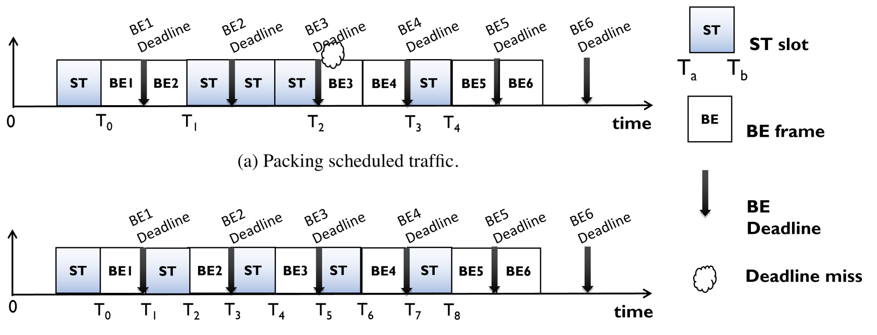


(b) Stream isolation.

Figure 2.4. Scheduling streams in TSN.

2.2.2 Heuristics Approaches

Many works in literature benefit from the flexibility of the heuristics algorithms for the sake of faster offline scheduling for ST class. This flexibility also allows for



(a) Packing scheduled traffic.

(a) Sparcifying scheduled traffic.

Figure 2.5. Example of schedule bin-packing and the effect on schedulability of BE traffic [21].

addressing more complex problem variations for different use cases of TSN. The authors in [23] consider multi-queuing in the offline scheduling. Considering the existence of multiple queues (all with the same priority) in the TSN egress port leads to increased schedulability of ST frames. To tackle the complexity of multi-queuing the algorithm in [23] schedules the ST streams subsequently per links in their route.

Reusch et al. in [24] point out sacrificing the frame-level network scheduling as the downside of asserting the formally defined frame and stream isolation constraints in the exact approaches, i.e., the isolation constraints in [19]. The authors [24] argue that in some use cases, the frame-level determinism of TTEthernet may still be required to be employed besides the stream-level determinism of TSN. Formally defining such requirements is challenging, hence the existing exact approaches for offline scheduling did not provide such flexibility. Accordingly, they propose a window-based heuristics that relaxes the isolation constraints.

The work by Gavrilut et al. [25] discusses that the network scheduling strategies for ST traffic can potentially increase the worst-case response times of the lower priority classes. This work proposes a two-stage heuristics for scheduling and routing ST. The objective of the solution is to simultaneously increase the schedulability of the traffic assigned to AVB class. Similarly, the authors in the work [26] synthesize offline schedules for TT traffic in TTEthernet with regards to the QoS of the RC traffic. Moreover, the algorithm proposed in [25] addresses the same problem for TSN traffic since it also accounts for the temporal isolation of the queues for deterministic ST schedules.

The work in [27] also aims at offline scheduling with considerations on improved schedulability of AVB class. The proposed AVB-aware network scheduling algorithm in [27] improves the work in [25] by considering the effect of preemption in the schedulability of the streams. In the first stage, the worst-case response times of the traffic assigned to the AVB class are calculated. Then, the response times of AVB streams are utilized in scheduling and choosing the best route for the ST streams.

Recently, a scheduling algorithm has been proposed in [28] that is intended for increasing the QoS of BE while increasing the pessimism in the schedules for the RC traffic in TTEthernet. The work in [29] presents a joint routing and scheduling algorithm along with two methods to reduce the number of back-jumps in the algorithm. These methods expedite the search for feasible solutions, therefore the algorithm is significantly more efficient than the previous related works, e.g., compared to [25]. Hence, it is claimed by the authors as suitable for solving scheduling problems for large-scale TSN networks.

Pahlevan et al. [30] propose a global heuristic list scheduler for scheduling tasks and ST streams for inter-task communication. One objective of this algorithm is to schedule ST frames back-to-back and inter-frame slacks to reduce the guard band overhead. Furthermore, in the work in [31], the same authors propose a Genetic Algorithm (GA)-based global scheduling. The motivation is faster network scheduling. The global heuristic list scheduler in [30] solves the network scheduling and routing problems independently from each other, therefore the generated schedules may over-utilize the links. Because the algorithm is agnostic of the link utilization, hence it assigns the frames to the first available link in the route to the destination end-station. In contrast, the proposed GA-based method in [31] performs the scheduling of the tasks and messages, and the routing of the inter-task communication depends on each other. Therefore, the method also optimizes the link utilization.

A recent similar work in [32] also employs GA for scheduling ST in TSN. The authors propose a genetic model for the ST frames, that can only be used for network scheduling to optimize end-to-end delays, jitter, and bandwidth utilization guard band. A recent GA-based approach is proposed in [33] that formulates a network scheduling objective function by reducing the schedule make-span (flow span) from the hyper period. The most suitable schedule is generated by maximizing the remaining bandwidth size. With this variation, the problem is still NP-hard. However, maximizing the remaining bandwidth and network utilization helps improve the QoS of the rest of the traffic classes in TSN, like the BE traffic. The authors further show improvement in the results with a GA-based approach compared to the optimization via an Integer Linear Programming (ILP)-based approach.

In the case of allowing multiple routes for the frames of the same stream, the arrival order of the frames to the destination can be affected. In the algorithm, proposed in [30], multi-cast transmission is supported by considering such paths as a set of uni-cast transmission paths. Consequently, the network scheduling is performed on each of those paths in an isolated manner. However, the multi-cast transmission model in [34] performs routing and scheduling of multi-path and forked streams (splitting streams) with the assumption that the destination node recovers out-of-order delivered frames.

Moreover, the work in [35] considers the redundancy, safety, and security requirements in large-scale TSN use cases. The authors in [35] propose two different solutions: firstly, a Constraint Programming (CP)-based routing and scheduling solution for small- to medium-scale problems, and secondly a two-phase solution that combines list scheduling and a Simulated Annealing (SA)-based algorithm for large-scale routing and scheduling problems. This work is the first work that embeds security constraints for scheduling ST streams.

Finally, [36] presents a hybrid task and message scheduling heuristic. The algorithm proposed in [36] schedules the tasks and messages separately in one phase. The next phase matches the task and message schedule in a way to optimizes the end-to-end data-propagation delays for the ST streams. The limitation of the proposed algorithm is that it only supports tasks with harmonic periods at the destination end-station. Though, outperforms the previous work with the exact approach [18] in terms of scheduling computation time.

2.3 Schedulability Analysis

This section presents the existing techniques for schedulability analysis of TSN networks. In general, there are four main techniques, including: (1) worst-case RTA; (2) Network Calculus (NC); (3) the concept of eligible intervals; and (4) TSN verification based on Machine Learning (ML) techniques. Figure 2.6 shows a timeline of the existing schedulability analysis approaches. According to [7], the focus of the first works on the existing schedulability analysis has evolved from only supporting CBS to more sophisticated models, which include the combination of CBS and TAS and frame preemption support. Moreover, the most recent works, e.g., [37], [38], and [39], include support for preemption in the schedulability analysis for networks based on TSN standards.

2.3.1 Worst-Case Response-Time Analysis (RTA)

The work in [40] is one of the first works targeting the application of Ethernet AVB for in-vehicle communication. The proposed RTA only supports CBS. In such a system model, traffic passing through class A is assumed to be the highest priority traffic. Hence, interference by any higher-priority traffic than class A is not considered.

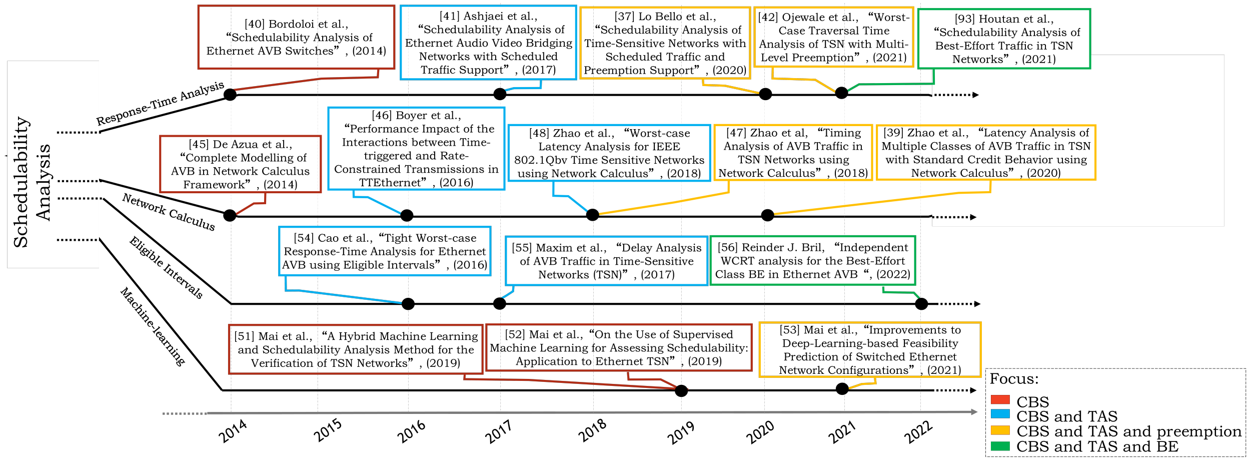

Schedulability Analysis
Response-Time Analysis
Network Calculus
Eligible Intervals
Machine-learning
[40] Bordoloi et al., "Schedulability Analysis of Ethernet AVB Switches", (2014)
[41] Ashjaei et al., "Schedulability Analysis of Ethernet Audio Video Bridging Networks with Scheduled Traffic Support", (2017)
[37] Lo Bello et al., "Schedulability Analysis of Time-Sensitive Networks with Scheduled Traffic and Preemption Support", (2020)
[42] Ojewale et al., "Worst-Case Traversal Time Analysis of TSN with Multi-Level Preemption", (2021)
[93] Houtan et al., "Schedulability Analysis of Best-Effort Traffic in TSN Networks", (2021)
[45] De Azua et al., "Complete Modelling of AVB in Network Calculus Framework", (2014)
[46] Boyer et al., "Performance Impact of the Interactions between Time-triggered and Rate-Constrained Transmissions in TTEthernet", (2016)
[48] Zhao et al., "Worst-case Latency Analysis for IEEE 802.1Qbv Time Sensitive Networks using Network Calculus", (2018)
[47] Zhao et al, "Timing Analysis of AVB Traffic in TSN Networks using Network Calculus", (2018)
[39] Zhao et al., "Latency Analysis of Multiple Classes of AVB Traffic in TSN with Standard Credit Behavior using Network Calculus", (2020)
[54] Cao et al., "Tight Worst-case Response-Time Analysis for Ethernet AVB using Eligible Intervals", (2016)
[55] Maxim et al., "Delay Analysis of AVB Traffic in Time-Sensitive Networks (TSN)", (2017)
[56] Reinder J. Bril, "Independent WCRT analysis for the Best-Effort Class BE in Ethernet AVB ", (2022)
[51] Mai et al., "A Hybrid Machine Learning and Schedulability Analysis Method for the Verification of TSN Networks", (2019)
[52] Mai et al., "On the Use of Supervised Machine Learning for Assessing Schedulability: Application to Ethernet TSN", (2019)
[53] Mai et al., "Improvements to Deep-Learning-based Feasibility Prediction of Switched Ethernet Network Configurations", (2021)
2014
2015
2016
2017
2018
2019
2020
2021
2022
Focus:
🟥 CBS
🟦 CBS and TAS
🟨 CBS and TAS and preemption
🟩 CBS and TAS and BE

Figure 2.6. Timeline of schedulability analysis techniques for TSN since 2014.

The work in [41] considered the CBS and TAS mechanisms in combination, where the TAS mechanism was a variation of the TSN standards. Furthermore, the work in [37] proposes a worst-case RTA that takes into account TAS, CBS, and different variations of frame preemption support (i.e., with and without Hold and Release mechanisms according to the TSN standard). Finally, the work in [42] introduced a worst-case traversal analysis, that is based on a prior technique that introduced a system model allowing multiple preemption levels [38].

The worst-case RTA is used in the work in [43] to calculate the minimum required bandwidth for CBS in a non-preemptive combination of TAS and CBS, i.e., in AVB ST [44]. AVB ST was proposed before the TSN standard amendments on the support for ST, thus the model and credit utilization were different than the TSN standards.

2.3.2 Network Calculus (NC)

Network calculus is a well-known technique to calculate the worst-case delays in networks. A network calculus analysis for AVB was introduced in [45]. Later, the work in [46] provided an approach to integrate the timing analysis for periodic TT traffic and RC sporadic traffic in TTEthernet. Although TTEthernet has several similarities to TSN, it does not feature the CBS mechanism. Thus, these analyses were not applicable to be used for TSN.

Within the context of TSN, the works in [47, 48] considered ST preempting AVB traffic. The focus of this work is to apply network calculus to calculate tighter worst-case bounds on delays for classes A and B in TSN. The work was further developed in [39] with an analysis based on network calculus for the case of having multiple AVB classes in the system model. It included the effects of traffic classes such as ST, A, and B. Besides, the work in [49] presented a solution that utilizes network calculus to include the credit behavior while generating ST schedules. Finally, recent work in [50] proposes a minimum bandwidth reservation technique for CBS classes based on NC. The technique proposed in [50] considers only CBS and excludes the effect of TAS mechanism.

2.3.3 Machine Learning (ML)

An approach for verifying the feasibility of TSN configurations by combining a schedulability analysis and an ML technique is presented in [51, 52]. The work in [53] further improved the approach by applying deep learning-based techniques. These techniques are not primarily proposed to provide predictability guarantees for TSN networks, yet they are interesting for verifying the TSN network configurations.

2.3.4 Eligible Intervals

Besides the aforementioned works, a technique based on the concept of eligible intervals was proposed in the context of AVB network, considering solely the CBS mechanism in [54]. Further, the work in [55] used the same concept to analyze the delays of classes A and B in the presence of the ST class. However, this technique was not used in more complex models in TSN timing analysis. Finally, the work in [56] proposes an RTA for the BE class. The analysis in [56] does not support preemption and considers AVB class as the highest priority class. Moreover, there are a few works that use eligible intervals, such as [57] and [58], to analyze the bandwidth reservation for AVB class in TSN networks. However, these works are limited to a single-switch architecture, and the effect of ST traffic on lower-priority traffic is neglected.

2.4 End-to-End Data-Propagation Delay Analysis

Embedded real-time systems are often modeled with chains of tasks and messages. To verify the timing behavior of these chains, not only their end-to-end response times need to be calculated and compared against the corresponding deadlines, but also the end-to-end data-propagation delays (data age and reaction time) should be calculated and compared with the corresponding data age and reaction time constraints. The timing constraints on the data age and reaction time delays are often specified on these distributed chains. The constraint on the data age delay is important, in particular, for control applications where the freshness of the data is of value. Whereas, the reaction time constraint is important in applications where the time of the first reaction to the input event is of value. These constraints are included in the timing model of the AUTOSAR standard [59] and are translated to several modeling languages in the vehicular domain [60].

2.4.1 Data-Propagation Delays in Single-Node Embedded Systems

To explain the data age and reaction time delays, consider a task chain consisting of three tasks τ_1, τ_2, and τ_3, as shown in Figure 2.7. All tasks belong to a single-core node and are activated independently. The periods of activation for tasks τ_1, τ_2, and τ_3 are 8 ms, 8 ms, and 4 ms, respectively. The Worst Case-Execution Time (WCET) of each task is assumed to be 1 ms. For simplicity, we assume that the priority of τ_1 is higher than the priority of τ_2 and the priority of τ_2 is higher than the priority of τ_3. By this priority assignment policy, we ensure that the precedent elements in the chain should be executed before their subsequent elements in the chain. The tasks use register-based communication, i.e., they communicate with each other and with their environment using writing data to/ and reading data from the registers. The registers are of non-consuming type. This means that data stays in the register after the reader has read the data. Furthermore, the registers are over-writable, i.e., if the writer is faster than the reader then the previous data in the register can be overwritten by the new data before the reader can read the previous data. The data read by τ_1 from Reg-1 corresponds to the input of the chain. Similarly, the data written to Reg-4 by τ_3 corresponds to the output of the chain.

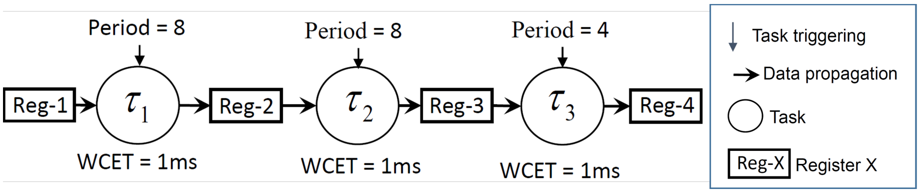


Figure 2.7. An example of a task chain that uses register-based communication.

As the tasks are activated independently and some tasks have different periods, the data traverses through the chain via multiple paths from the input to the output of the chain as shown in Figure 2.8. These paths are called timed-paths (also referred to as data-paths). Due to multiple timed-paths, there can be various delays in delivering the data from the input to the output of the chain.

The data age delay is the time elapsed between the arrival of data at the input and the latest availability of the corresponding data at the output. In the data age delay analysis, we are interested in identifying the longest time difference between the input data and the last sample of corresponding output data. On the other hand, the reaction time delay corresponds to the earliest availability of the data at the first instance of the output corresponding to the data that *just missed* the read access at the input. An event (corresponding to the availability of data) is considered readable by an instance of a task if it occurs at or before the activation of the task. If the event happens just after the activation of the task instance, the data is not readable to this instance, i.e., the data is just missed by the current instance of the task. The missed data is read by the next instance of the task. This is illustrated by the white thunderbolt in Figure 2.8, where the first instance of τ_1 at time 0 misses the data but the same data is read by the next instance of τ_1 at time 8.

Possible data age and reaction time delays in the chain in Figure 2.7 are shown in Figure 2.8. On the one hand, the data from the event happening a bit before time 16 is accessible to the third instance of τ_1 (activated at time 16). In such a case, the latest impact of this event is available at the output of the chain until 5 ms after the occurrence of the event (data age delay). On the other hand, the sampling of the data coming from the event happening a bit after the time 0 is delayed until the time 8, where the data can be read by the second instance of τ_1. Accordingly, the earliest time the impact of the data appears at the output of the chain is 11 ms after the occurrence of the event (reaction time delay).

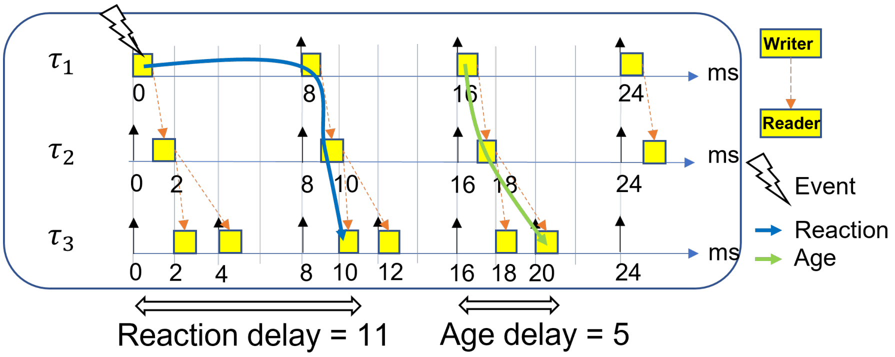


Figure 2.8. Data age and reaction time delays in the task chain are depicted in Figure 2.7.

2.4.2 Data-propagation Delays in Distributed Embedded Systems

The data-propagation delays are equally valid in distributed embedded systems. Let us consider a distributed task chain in a distributed embedded system depicted in Figure 2.9, where two nodes are connected via a network. In this example, the tasks are activated periodically with periods of 6 ms and 3 ms, respectively. Task τ_1 in Node 1 sends a message to task τ_2 in Node 2 through the network.

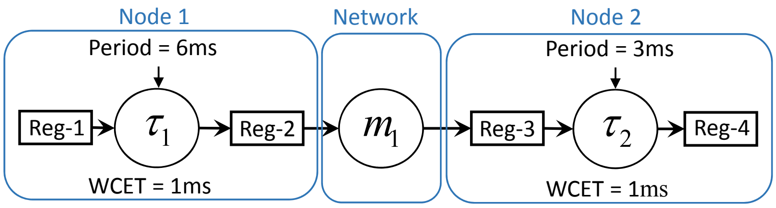


Figure 2.9. A multi-rate chain in a distributed embedded system.

Depending on the type of network, we may have different possible timed-paths through which the data can propagate from the sender task to the receiver task. For example, when the network is not capable of initiating communication independent of the sending tasks, a message can only be queued for transmission at the network interface by the sending task. This is the case for many event-triggered network protocols, like CAN [1]. In this case, the message inherits its period from the sender task. Furthermore, the timed-paths in a distributed task chain also depend upon whether the network supports the synchronization of nodes. For example, TSN supports synchronization among the end-stations via the IEEE 802.1AS standard, whereas the CAN protocol does not support synchronization. Figure 2.10 shows an execution trace when the nodes are synchronized in the system that is shown in Figure 2.9. The data age and reaction time delays in this distributed chain are identified as 7 ms and 10 ms, respectively.

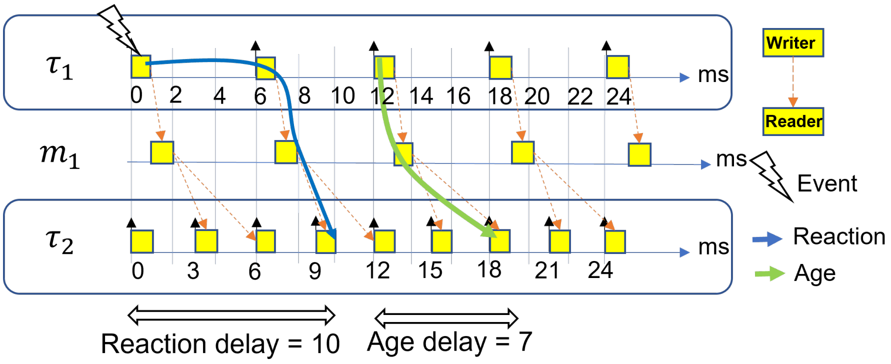
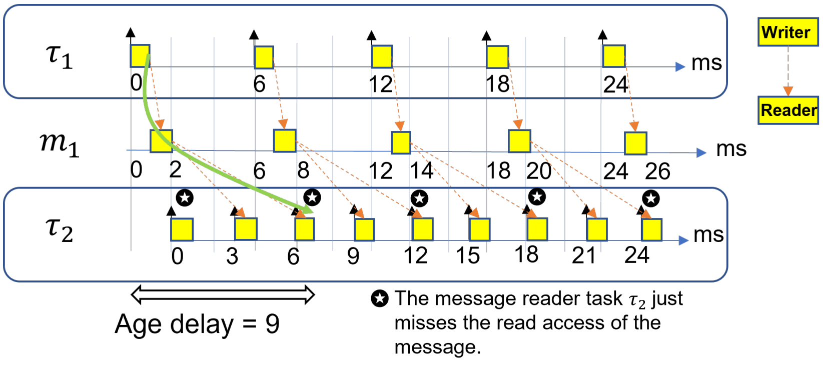


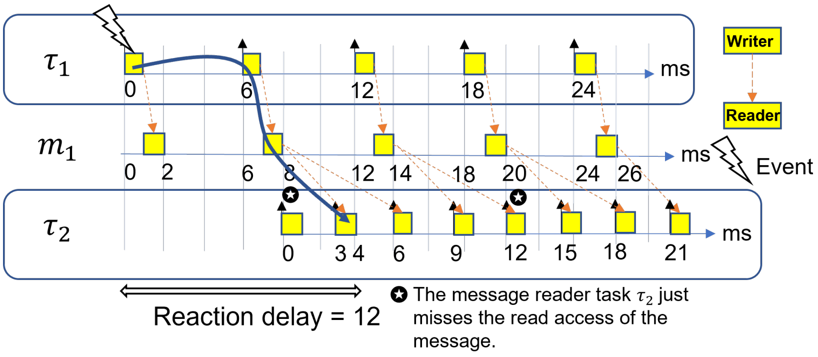
Figure 2.10. A possible execution trace for the distributed embedded system example is shown in Figure 2.9 when source and destination end-stations are synchronized.

A possible execution trace of the distributed task chain in Figure 2.9 when the nodes are not synchronized is shown in Figure 2.11(a). To create worst-case conditions when the nodes are not synchronized, we assume that the receiver task τ_2 is activated "just before" the arrival of the message at the receiver node. Hence, the current instance of τ_2 (the first period activation) will miss the read access of the message. The message will be read by the next instance of τ_2 (second period activation) as shown in Figure 2.11(a). The corresponding data age delay is identified as 9 ms as shown in Figure 2.11(a).

To increase readability, we draw the same execution trace separately for the case of reaction time delay in the distributed task chain (as shown in Figure 2.9) when the nodes are not synchronized as depicted in Figure 2.11(b). In Figure 2.11(b), the first instance of τ_1 is activating the first instance of the message m_1. According to the assumption for the reaction time delay, the first instance of τ_1 has missed the sampling of the chain's input event, thus the first instance of m_1 does not deliver valid data from the input event to the receiver task. However, as the second instance of τ_1 reads the input event, the second instance of the message m_1 also holds fresh data. Since τ_2 is not synchronized with τ_1, the worst-case assumption is that τ_2 is activated a small amount of time earlier than the arrival of the message that holds the sampled data. Consequently, the first instance of τ_2 misses to read data of the event, which is being written by the second instance of m_1. But, at the next instance of τ_2 (second period activation), τ_2 can read the incoming data from m_1. Accordingly, the reaction time delay is 12 ms.



(a) Data age delay.



(b) Reaction time delay.

Figure 2.11. A possible execution trace for the distributed embedded system example is shown in Figure 2.9 when source and destination end-stations are not synchronized.

According to the classification in the work in [61], there are three approaches for calculating the worst-case end-to-end data-propagation delays: 1) simulation-based method that obtains the maximum end-to-end delays from a set of selected scenarios. The simulation approach does not necessarily show the maximum end-to-end delay, because it might not cover all the possible assumptions for the worst-case scenario; 2) model-checking based on an exhaustive search can provide exact worst-case end-to-end delays even on large-scale networks. However, these methods have high time complexity; and 3) an analytical approach that provides upper bounds on end-to-end delays with a certain pessimism. The focus of this thesis is on the group of analytical approaches in the above-mentioned classification.

From a different perspective, the work in [62] mentions active and passive approaches for the end-to-end data-propagation delay analysis. Active approaches optimize the pattern of task releases in a chain to achieve optimal delays. Whereas, passive approaches study the worst-case assumptions for the end-to-end delays in distributed embedded systems to find upper bounds on the delays.

The existing end-to-end data-propagation delay analysis can incorporate the response-time analysis of various legacy real-time networks, such as CAN [63], FlexRay, legacy Ethernet [64], or TSN. In the following paragraphs, we present various existing end-to-end data-propagation delay analysis approaches that are related to the work in the thesis.

Active Approaches The set of works in this category focuses on different network design and configuration strategies that lead to optimized end-to-end data-propagation delay analysis. Some design parameters include periods and offsets. Active approaches are similar to system-level off-line scheduling approaches, though the scope of these approaches also covers optimizing the end-to-end data-propagation delays of the non-real-time traffic.

The work in [65] targets data age delay in cause-effect chains within one execution node. The tasks are synchronized inside an end-station, and they are scheduled with offsets. The aim of the work in [65] is to find priorities, and offsets and to optimize the design mapping for tasks to minimize the data age delays in a chain of tasks. Similarly, the work in [66] aims at finding offsets for the chain of tasks to optimize the end-to-end delays. The works in [65, 66] belong to the active end-to-end data-propagation analysis. Moreover, the work in [67] studies the dependencies between the task instances. Such dependencies can be specified at early development stages to guarantee data age delay constraint.

Passive Approaches The work in [62] performs end-to-end timing analysis for the systems with locally synchronized periodic tasks in one end-stations. The end-stations only support non-synchronized communication, i.e., via CAN, or FlexRay. Similarly, the work in [68] considers globally non-synchronized communication among the end-stations, while the tasks within the end-stations are considered synchronized. The authors in [68] propose a computationally efficient analysis compared to the analysis in [62]. Besides, the work in [68] achieves a higher upper bound on the data age delay than the work in [62].

The focus of the aforementioned works is only on data age delay, whereas our analysis also includes the analysis of the reaction time delay. The work in [69] is a seminal work in the literature that introduces a formal framework for defining end-to-end delays in periodic and register-based systems. The task model in [69] is based on the Bounded Execution Time (BET) task model. In the BET task model, the communication between the tasks is implicit where a task reads data from the register at its beginning and writes to the register at its end of execution. Furthermore, the work in [70] proposes an end-to-end data-propagation delay analysis that includes sporadic tasks based on the BET task model. The work in [71] builds on the Logical Execution Time (LET) paradigm by proposing a system-level LET-based communication model for distributed embedded systems. LET is an inter-task communication model that augments the read/write access times (input and output of the task) to the task's physical execution time model. System-level LET models the communication between tasks of different end-stations. In the work presented in [71], the end-stations have their local clocks (timeline). The global timeline is approximated based on the local timeline of the sender and receiver end-stations with a bounded error. In our end-to-end data-propagation delay analysis, we rely on the determinism promised for ST traffic, and exclude the global synchronization error for analyzing transactions that utilize ST traffic. The work in [72] considers globally non-synchronized and locally synchronized task chains and proposes a method to calculate a limited number of timed-paths[2] that lead to the maximum end-to-end delays.

While the majority of the works focus on providing methods for calculating the maximum end-to-end delays in the chains, the work in [73] discusses robustness margins around the end-to-end timing constraints. The work in [73] employs BET and system-level LET communication model, and further studies the variations in the tasks' response times and the influence on the robustness of the system (i.e., variations in the end-to-end delays).

The work in [74] proposes an end-to-end data-propagation delay analysis for the chains of tasks in the context of Robot Operating System (ROS). The analysis before [74] mainly focused on analyzing periodic and sporadic tasks. The work in [74] extends the end-to-end timing analysis to support ROS2 task chains that deal with a mix of

[2]We use the terms timed-paths and data-paths interchangeably to refer to the path that the data traverses from one task to another within the task chain.

time-triggered and event-triggered functions. Besides, the network model in [74] is based on the publisher-subscriber communication model. The communication model is therefore inherently non-synchronized.

The existing timed-path calculation used in the end-to-end data-propagation delay analysis algorithms, such as [69, 72], applies to traffic classes that do not require offline schedules, i.e., AVB and BE. For example, the work in [72] provides end-to-end analysis for the synchronized communication between sender and receiver end-stations, but it only considers the case when non-ST are transmitted between synchronized end-stations, i.e. AVB or BE.

Though the BET and LET models can be mapped to each other according to [73]. In this thesis, we chose to use the BET model because it is already integrated into several tools (including industrial tools) that support model- and component-based software development of vehicular embedded systems, e.g., in [75, 76].

2.5 Software Architecture Modelling

There are several software architecture modeling languages and component models, such as AUTOSAR, RCM, AMALTHEA, EAST-ADL, AADL, to mention a few.

2.5.1 AUTomotive Open System ARchitecture (AUTOSAR)

AUTOSAR is widely used for developing software architecture for automotive systems. SymTA/S [77] is one of the commercial timing analysis and optimization toolchains that complies with AUTOSAR. A recent work in [78] integrates AUTOSAR adaptive with applicable standards to develop more sophisticated systems. The proposed three-layer architecture coordinates a binding between AUTOSAR Adaptive, OPC Unified Architecture (OPC UA) standards [79], and TSN standards. Notnetheless, the work in [78] argues that the proposed architecture in later work lacks maturity since all the involved technologies in the architecture layers are still under development.

2.5.2 RUBUS Component Model (RCM)

According to the work in [80], RCM is the architectural language supporting the largest number of modelling elements required by centralised architectures than the other software architecture modeling languages. Besides, as evaluated in [81, 82], RCM comprises hierarchical entities that are necessary to model a distributed embedded system that supports TSN. At the highest level, the system contains at least two nodes and a network element that interconnects the nodes. A node (end-station) is a processing element that provides a run-time environment for one or more Software Applications.

In RCM, a software architecture is modeled by interconnecting a set of SWC. An SWC is a design-time entity that may correspond to a task at run-time or in the timing model. The SWCs communicate with each other by their interfaces (a set of data and trigger ports). To the best of our knowledge, RCM is the first and only component model that supports comprehensive modeling of TSN. Recently, there have been some efforts in increasing the performance and applicability of the other modeling approaches to RCM by model transformation [83]. The work in [84] proposes a mapping technique between AMALTHEA and RCM to enable timing analysis of AMALTHEA-based component models in RCM. In addition, the work in [85] presents a mapping from EAST-ADL models to RCM to enable the timing analysis of a non-RCM model.

2.5.3 Simulation of Vehicular Systems

Besides the analytical schedulability analysis techniques, simulation and modeling of TSN networks can also contribute to more efficient configuration design and scheduling of TSN networks [7]. OMNeT++ is an open-source discrete event simulation platform, primarily used for constructing simulations of networks[3]. INET[4] is another widely used open-source simulation framework that models wired, wireless, and mobile networks. There are two most commonly used simulation frameworks for TSN built on OMNeT++: (i) Core4INET [86] (ii) NeSTiNg [87]. Firstly, Core4INET allows the simulation of TSN networks based on various standards, e.g., IEEE 802.1Q, IEEE 802.1p VLANs and Priorities, IEEE 802.1 AVB, and TTEthernet (AS6802). Secondly, NeSTiNg supports several TSN features, including the ST (IEEE 802.1Qbv), frame preemption (IEEE Std 802.1Qbu and IEEE Std 802.3br), credit-based shaper (IEEE Std 802.1Qav), and clock synchronization (IEEE Std 802.1AS). However, the aforementioned simulation frameworks are ongoing projects, and the TSN features are partially implemented in them.

[3]https://omnetpp.org/

[4]https://inet.omnetpp.org/

Chapter 3

Research Contributions

In this chapter, we briefly present the three Thesis Contributions (TCs). Then, we present the mapping between the RGs, the TCs, and the included publications in this thesis. Finally, a summary of each of the included papers is presented.

3.1 Thesis Contributions

3.1.1 TC$_1$: End-to-end Timing Analysis of TSN-based Systems

We extend the existing worst-case RTA [37] for TSN to support the BE traffic class. Thanks to the extended RTA, we can take the second step in TC$_1$ and develop an end-to-end data-propagation delay analysis that supports synchronization as well as non-synchronization among the end-stations. The proposed end-to-end data-propagation delay analysis is built on an existing analysis [69, 88] and extends the time-path calculation algorithm to support the analysis of the systems where the end-stations connected to the TSN network are synchronized. The proposed analysis is backward compatible with the analysis of all non-scheduled traffic (non-ST) classes (AVB or BE) when the end-stations are not synchronized in the TSN networks.

The purpose of the developed techniques within TC$_1$ is to address the challenges that are faced in achieving the research goal RG$_1$.

3.1.2 TC$_2$: Configuration of Various Traffic Classes in TSN

Firstly, we propose a modular framework as shown in Figure 3.1 for the automatic configuration of ST. The proposed framework consists of four main modules, namely 1) traffic generator (Figure 3.1-module 1); 2) schedule synthesizer (Figure 3.1-module 2); 3) automated configurator and Graphical User Interface (GUI) (Figure 3.1-module 3); and 4) results interpretation (Figure 3.1-module 4). The proposed framework integrates the TSN schedule synthesizer with a widely used TSN simulation tool (NeSTiNg plug-in [87, 89]). Besides, it allows to automatically generate and visualize schedules for TSN switches, i.e., visualizing TSN flows, GCL, and the gate states at an egress port.

Secondly, we propose a technique to generate optimized schedules for ST in TSN while considering the QoS for the BE traffic. The proposed technique comprises scheduling constraints for a CP-based scheduler and various objective functions to generate ST schedules with different patterns, i.e., packing schedules by minimization or maximization of offsets, or generating sparse schedules.

Thirdly, we develop a solution to analyze the required bandwidth for AVB traffic considering features such as multi-hop architecture, impact of ST traffic, and preemption support of ST over lower-priority traffic classes. The proposed solution analyzes the required bandwidth according to the widely adopted worst-case RTA and calculates the minimum credit which makes all traffic assigned as AVB schedulable.

The purpose of the developed techniques within TC$_2$ is to address the challenges that are faced in achieving the research goal RG$_2$.

3.1.3 TC$_3$: Timing Analysis of Component-based Distributed Software Architectures

In TC$_3$, we propose an automated methodology for extracting the end-to-end timing model from component-based software architectures of TSN-based embedded systems. In particular, to generate an end-to-end timing model, we have identified various sources of end-to-end timing information within the development process of TSN-based systems. These sources include: 1) user (i.e., software architecture developer); 2) software architectures; 3) system configuration; and 4) end-to-end timing analysis.

The proposed methodology within the system and software modeling stage is shown in Figure 3.2. The methodology is aimed at extracting the end-to-end timing information and populate them in en end-to-end timing model within module c as shown in Figure 3.2. In this methodology, the component model description of the system (Figure 3.2-module b1) and the network configuration (Figure 3.2-module b2) are the

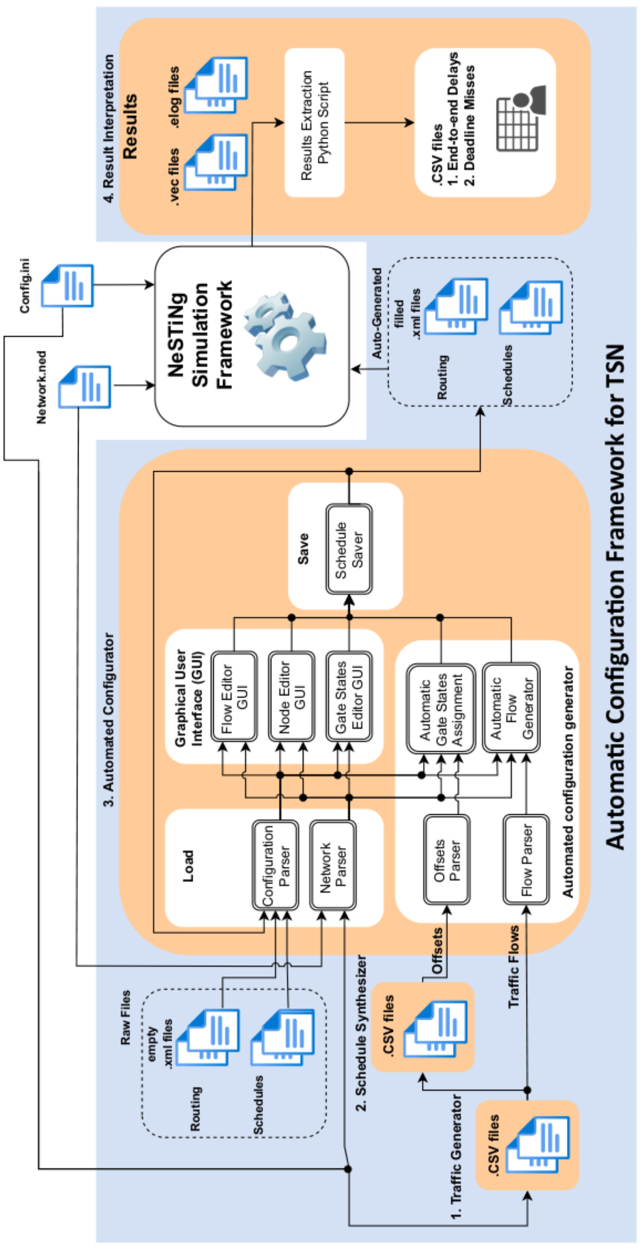


Figure 3.1. Automatic configuration framework for TSN [90].

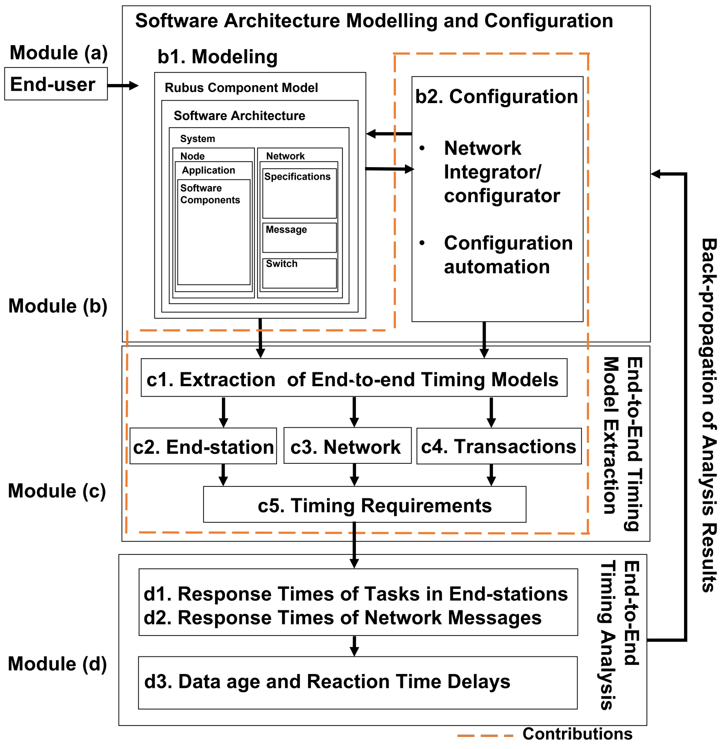

Software Architecture Modelling and Configuration
Module (a)
End-user
b1. Modeling
Rubus Component Model
Software Architecture
System
Node
Application
Software Components
Network
Specifications
Message
Switch
b2. Configuration
• Network Integrator/ configurator
• Configuration automation
Module (b)
c1. Extraction of End-to-end Timing Models
c2. End-station
c3. Network
c4. Transactions
c5. Timing Requirements
Module (c)
End-to-End Timing Model Extraction
d1. Response Times of Tasks in End-stations
d2. Response Times of Network Messages
d3. Data age and Reaction Time Delays
Module (d)
End-to-End Timing Analysis
Back-propagation of Analysis Results
Contributions

Figure 3.2. Timing model extraction methodology [91].

sources of timing information. Accordingly, the timing information is extracted from the appropriate sources which we identified. Then, the end-to-end timing models are provided as input to the end-to-end timing analysis stage (Figure 3.2-module d).

The purpose of the developed techniques within TC$_3$ is to address the challenges that are faced in achieving the research goal RG$_3$.

3.2 Research Papers

The TCs are encapsulated in six publications: five conference papers and one journal article. Table 3.1 presents the mapping of the included papers, RGs, and TCs. This section presents a summary of the publications included in this thesis.

	RG_1	RG_2	RG_3
Paper A		TC_2	
Paper B		TC_2	
Paper C	TC_1		
Paper D	TC_1		
Paper E			TC_3
Paper F		TC_2	

Table 3.1. Mapping of the publications, Research Goals (RGs) and the Thesis Contributions (TCs).

3.2.1 Paper A: An Automated Configuration Framework for TSN Networks

Title: An Automated Configuration Framework for IEEE 802.1 Time Sensitive Networking [90]
Authors: Bahar Houtan, Albert Bergström, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Saad Mubeen.
Status: Published in the IEEE 22^{nd} International Conference on Industrial Technology (ICIT), 2021.

Abstract:

Designing and simulating large networks, based on the Time-Sensitive Networking (TSN) standards, require complex and demanding configuration at the design and pre-simulation phases. Existing configuration and simulation frameworks support only the manual configuration of TSN networks. This hampers the applicability of these frameworks to large-sized TSN networks, especially in complex industrial embedded system applications. This paper proposes a modular framework to automate offline scheduling in TSN networks to facilitate the design-time and pre-simulation automated network configurations as well as interpretation of the simulations. To demonstrate and evaluate the applicability of the proposed framework, a large TSN network is automatically configured, and its performance is evaluated by measuring end-to-end delays of

time-critical flows in a state-of-the-art simulation framework, namely NeSTiNg.

Contributions:

- Integrating the traffic generation and network schedule synthesis to a TSN open-source simulation framework, i.e., NeSTiNg;
- Addressing the complexity of flow configuration by automatically translating configuration for the ST into the syntax compliant with the NeSTiNg simulation framework;
- Addressing the complexity of gate state configuration by automatically translating the synthesized ST schedules into the syntax compliant with NeSTiNg simulation framework; and
- Automatic generation and modification of the configuration files by a GUI that is integrated into the NeSTiNg simulation framework.

Authors' Contributions:
Part of this work was initiated as a master thesis work, in which Albert Bergström was supervised by me and Mohammad Ashjaei. The work in the master thesis was used as the basis for the publication and then further reworked and developed by me, supervised by the rest of the co-authors. I performed the tool evaluations and wrote the draft of the paper. The co-authors reviewed the paper, after which I improved it.

3.2.2 Paper B: Synthesising Schedules to Improve QoS of Best-effort Traffic in TSN Networks

Title: Synthesising Schedules to Improve QoS of Best-effort Traffic in TSN Networks [21]
Authors: Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Saad Mubeen.
Status: Published in the 29th International Conference on Real-Time Networks and Systems (RTNS), 2021.

Abstract:
The IEEE Time-Sensitive Networking (TSN) standards' amendment 802.1Qbv provides real-time guarantees for Scheduled Traffic (ST) streams by the Time-Aware Shaper (TAS) mechanism. In this paper, we develop offline schedule optimization objective functions to configure the TAS for ST streams, which can be effective to

achieve a high Quality of Service (QoS) of lower priority Best-Effort (BE) traffic. This becomes useful if real-time streams from legacy protocols are configured to be carried by the BE class or if the BE class is used for value-added (but non-critical) services. We present three alternative objective functions, namely Maximization, Sparse and Evenly Sparse, followed by a set of constraints on ST streams. Based on simulated stream traces in OMNeT++/INET TSN NeSTiNg simulator, we compare our proposed schemes with a most commonly applied objective function in terms of overall maximum end-to-end delay and deadline misses of BE streams. The results confirm that changing the schedule synthesis objective to our proposed schemes ensures timely delivery and lower end-to-end delays in BE streams.

Contributions:

- Mathematically modeling and presenting optimization constraints to consider the QoS for the BE traffic;
- Presenting new optimization objective functions to obtain a feasible ST schedule while improving the QoS of the BE traffic;
- Comparing the commonly used objective function (minimizing the ST offsets) with the proposed objective functions, in terms of affecting the QoS for any lower priority traffic class;
- Evaluating the proposed solutions using an OMNeT++ simulation platform.

Authors' Contributions:
I was the main driver of the work under the supervision of the co-authors. The plan for the paper was formed in joint discussions with the co-authors. I performed the tool evaluations and wrote the draft of the paper. The co-authors reviewed the paper, after which I improved it.

3.2.3 Paper C: Schedulability Analysis of Best-Effort Traffic in TSN Networks

Title: Schedulability Analysis of Best-Effort Traffic in TSN Networks [92]
Authors: Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Sara Afshar, Saad Mubeen.
Status: Published in the IEEE 26^{th} International Conference on Emerging Technologies

and Factory Automation (ETFA), 2021.

Abstract:

This paper presents a schedulability analysis for the Best-Effort (BE) traffic class within Time-Sensitive Networking (TSN) networks. The presented analysis considers several features in the TSN standards, including the Credit-Based Shaping (CBS), the Time-Aware Shaper (TAS) and the frame preemption. Although the BE class in TSN is primarily used for the traffic with no strict timing requirements, some industrial applications prefer to utilize this class for the non-hard real-time traffic instead of classes that use the CBS. The reason mainly lies in the fact that the complexity of TSN configuration becomes significantly high when the time-triggered traffic via the TAS and other classes via the CBS are used altogether. We demonstrate the applicability of the presented analysis on a vehicular application use case. We show that a network designer can get information on the schedulability of the BE traffic, based on which the network configuration can be further refined with respect to the application requirements.

Contributions:

- A schedulability analysis to verify the worst-case response time of each individual BE message in the network when CBS, TAS, and frame preemption are used.

Authors' Contributions:

I was the main driver of the work under the supervision of the co-authors and Sara Afshar who provided me with feedback as the representative of our industrial partners. The plan for the paper was formed in joint discussions with the co-authors. I performed the tool evaluations and wrote the draft of the paper. The co-authors reviewed the paper, after which I improved it.

3.2.4 Paper D: Supporting End-to-end Data-Propagation Delay Analysis for TSN-based Distributed Vehicular Embedded Systems

Title: Supporting End-to-end Data-Propagation Delay Analysis for TSN-based Distributed Vehicular Embedded Systems [93]
Authors: Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Saad Mubeen
Status: Published in the Journal of Systems Architecture (JSA), 2023.

Abstract:

In this paper, we identify that the existing end-to-end data-propagation delay analysis for distributed embedded systems can calculate pessimistic (over-estimated) analysis results when the nodes are synchronized. This is particularly the case of the Scheduled Traffic (ST) class in Time-Sensitive Networking (TSN), which is scheduled offline according to the IEEE 802.1Qbv standard, and the nodes are synchronized according to the IEEE Std 802.1AS standard. We present a comprehensive system model for distributed embedded systems that incorporates all of the above-mentioned aspects as well as all traffic classes in TSN. We extend the analysis to support both synchronization and non-synchronization among the end-stations as well as offline schedules on the networks. The extended analysis can now be used to analyze all traffic classes in TSN when the nodes are synchronized without introducing any pessimism in the analysis results. We evaluate the proposed model and the extended analysis on a vehicular industrial use case.

Contributions:

- We extend the timed-path computation algorithm within the existing end-to-end data-propagation delay analysis to support all traffic classes in TSN networks when the end-stations are synchronized using the IEEE Std 802.1AS standard. Unlike the existing algorithm, the analysis results with the extended algorithm do not include any pessimism when the end-stations in the TSN networks are synchronized. The extended algorithm is backward compatible to support the analysis of all non-ST classes[1] (A, B, BE) when the end-stations are not synchronized in the TSN networks.

- We present a comprehensive system model for distributed embedded systems to support the extended algorithm, which incorporates all traffic classes in TSN. The model can express distributed task chains that can contain various types of traffic supported by TSN, including the ST, A, B, and BE traffic.

- We demonstrate the applicability of the presented model and analysis to a vehicular industrial use case. We also perform a comparative evaluation of the extended analysis with the existing analysis by analyzing the use case with the two analyses. Furthermore, the presented model and analysis are evaluated by experiments to show the effect of various configurations of ST class, receiver periods, and synchronization of the sender and receiver end-stations on the end-to-end data-propagation delays.

[1]It is required to use synchronization when the ST traffic class in TSN is used.

Authors' Contributions:

I was the main driver of the work under the supervision of the co-authors. The plan for the paper was formed in joint discussions with the co-authors. I performed the tool evaluations and wrote the draft of the paper. The co-authors reviewed the paper, after which I improved it.

3.2.5 Paper E: End-to-end Timing Modeling and Analysis of TSN in Component-Based Vehicular Software

Title: End-to-end Timing Modeling and Analysis of TSN in Component-Based Vehicular Software [91]
Authors: Bahar Houtan, Mehmet Onur Aybek, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, John Lundbäck, Saad Mubeen
Status: Published in the IEEE 26th International Symposium On Real-Time Distributed Computing (ISORC), 2023.

Abstract:

In this paper, we present an end-to-end timing model to capture timing information from software architectures of distributed embedded systems that use network communication based on Time-Sensitive Networking (TSN) standards. Such a model is required as an input to perform end-to-end timing analysis of these systems. Furthermore, we present a methodology that aims at automated extraction of instances of the end-to-end timing model from component-based software architectures of the systems and the TSN network configurations. As a proof of concept, we implement the proposed end-to-end timing model and the extraction methodology in the RUBUS Component Model (RCM) and its toolchain Rubus-ICE that are used in the vehicle industry. We demonstrate the usability of the proposed model and methodology by modeling a vehicular industrial use case and performing its timing analysis.

Contributions:

- We propose an end-to-end timing model to describe TSN networks with all configuration parameters in the distributed vehicular embedded software systems.
- We provide an automated methodology to extract instances of the end-to-end timing model from component-based software architectures of vehicular systems.

- We provide a proof of concept for the proposed timing model and extraction methodology by integrating them with the component-based software engineering environment of an industrial tool suite, namely Rubus-ICE.
- We evaluate the proposed model and methodology as well as their integration with the Rubus-ICE tool suite on a vehicular industrial use case.

Authors' Contributions:

I was the main driver of the work under the supervision of the co-authors, Mehmet Onur Aybek and John Lundbäck as representatives of our industrial partners. The plan for the paper is formed in joint discussions with the co-authors. I performed the tool evaluations and wrote the draft of the paper. The co-authors reviewed the paper, after which I improved it.

3.2.6 Paper F: Bandwidth Reservation Analysis for Schedulability of AVB Traffic in TSN

Title: Bandwidth Reservation Analysis for Schedulability of AVB Traffic in TSN [94]
Authors: Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Saad Mubeen
Status: Accepted for publication in the IEEE 25th International Conference on Industrial Technology (ICIT), 2024.

Abstract:

In this paper, we present a Bandwidth Reservation Analysis (BRA) for Audio-Video Bridging (AVB) traffic in the Time-Sensitive Networking (TSN) standards. The proposed analysis is based on the existing worst-case Response-Time Analysis (RTA) and can be used to calculate the minimum required bandwidth for guaranteeing the schedulability of the messages in AVB classes. The proposed analysis is applicable for allocating sufficient bandwidth for the schedulability of AVB traffic in case of using a combination of Time-Aware Shaper (TAS) and Credit-Based Shaping (CBS) mechanisms and takes into account the interference by Scheduled Traffic (ST) preemption and overhead. Besides, the proposed BRA is applicable for allocating bandwidth per link in the route of the AVB traffic according to the deadlines per link of traffic. We present an evaluation based on an automotive use case. We evaluate the schedulability of AVB traffic by comparing the proposed BRA with the utilization-based bandwidth reservation as recommended by the TSN standards.

Contributions:

- In this paper, we propose a Bandwidth Reservation Analysis (BRA) for calculating the lowest value of the bandwidth for classes A and B such that they become schedulable. The solution is based on the existing worst-case RTA to consider the impact of ST traffic, multi-hop architecture, and preemption of ST on the lower-priority traffic.

Authors' Contributions:
I was the main driver of the work under the supervision of the co-authors. The plan for the paper was formed in joint discussions with the co-authors. I performed the tool evaluations and wrote the draft of the paper. The co-authors reviewed the paper, after which I improved it.

Chapter 4

Discussions

In this chapter, we first assess the extent to which the TCs address the challenges to the RGs. We further discuss the results and limitations that we faced in the research process.

4.1 Connecting Thesis Contributions to the Research Goals

RG$_1$: Develop an end-to-end data-propagation delay analysis for TSN-based distributed embedded systems. In our studies, we have perceived that the general concern of the research on schedulability analysis has been on the worst-case scenarios for analyzing the ST and AVB classes which subsequently undergo the TAS and CBS mechanisms. These classes come with TSN mechanisms to support preserving the timing requirements of the traffic class that they are intended for. However, in practice, the TSN standards still allow the designers to assign real-time traffic to the lower priority traffic class, i.e. BE class.

We observed that the flexibility of choosing the BE class gives the developers free hands for increasing the system's functionalities. The benefits of this flexibility are: firstly, legacy real-time systems without TSN support can be easily integrated to TSN via BE class; secondly, the workload of configuration can be compensated by assigning some real-time traffic with less strict timing requirements to BE class. For example, BE can be used when one of the higher priority classes is overloaded in an application, or when the designers face a high configuration workload in case multiple traffic shaping mechanisms are used; thirdly, new traffic can be included in the system simply by assigning them to go through BE class. As a result, there is no need to change the configuration of the shaping mechanisms that have been set to guarantee the schedulability of the already existing traffic.

Our first step towards reaching $\mathbf{RG_1}$ was to address the challenge of incorporating TSN classes in the worst-case RTA (RG_1-challenge$_1$). We particularly aimed to support the QoS of BE by verifying its timing requirements at the timing analysis stage.

To the extent of our knowledge, the proposed worst-case RTA for BE was the only analysis, among the few existing, that incorporated various TSN classes and mechanisms, i.e., for classes ST, A and B via the TAS and CBS, and the frame preemption mechanisms. Nevertheless, due to limitations of the existing analysis approaches, i.e. lack of support for BE or limitations in supporting TSN mechanisms, we could not perform a comparative evaluation of the proposed analysis. Thus our use case evaluation showed the applicability of the proposed analysis for calculating the worst-case response times of the frames in BE class. Moreover, we have integrated the proposed RTA for BE in an industrial tool suite, namely Rubus Integrated Component model development Environment (ICE) which has undergone evaluation using several industrial use cases.

However, it is still challenging to guarantee the schedulability of BE traffic. Because of the lack of a shaping mechanism, there can be no guarantee for the schedulability of the BE traffic within the development process. This raises the need for further supporting the QoS of BE via the configuration stage, e.g., leaving sufficient bandwidth for traffic assigned to BE classes via properly configuring the shaper mechanisms intended for the higher priority traffic. This problem is later studied in $\mathbf{RG_2}$.

In our next step towards $\mathbf{RG_1}$, we expanded our perspective towards incorporating various TSN mechanisms within the end-to-end data-propagation delay analysis which was the second identified challenge towards this RG (RG_1-challenge$_2$). Throughout our literature reviews, we noticed that the majority of the existing works only considered local synchronization with a local notion of time within the end-stations. The synchronization mechanisms defined in IEEE 802.1AS enable global synchronization of the sender and receiver end-stations. Moreover, TSN traffic classes have different synchronization requirements, e.g., ST requires synchronization for deterministic transmission of frames via TAS mechanism. However, synchronization mechanisms are optional for other traffic classes and their need for synchronization of end-stations depends on the application requirements.

In our proposed end-to-end data-propagation delay analysis, we elaborate on different worst-case assumptions for calculating the end-to-end delays in the case where end-stations are globally synchronized. In particular, we extended the timed-path calculation algorithm to remove the pessimistic assumption of the existing analysis which only supported the non-synchronized end-stations. It is worth mentioning that we kept the proposed analysis backward compatible to let it remain applicable to analyzing non-synchronized end-stations.

The proposed analysis further allowed us to demonstrate the effect of various configurations of TAS mechanism for ST class, receiver task periods, and synchronization of the sender and receiver end-stations on the end-to-end data-propagation delays of transactions dealing with ST class. In particular, the results of the proposed analysis showed that the configuration of TAS has a great impact on the end-to-end data-propagation delays of traffic passing through ST class. We deem the aforementioned insight significant since the existing end-to-end data-propagation analysis was not applicable for calculating end-to-end delays for the transactions that utilize ST class. This insight shows the applicability of the proposed analysis for further research on the configuration of TSN systems.

RG_2: Develop network configuration techniques considering the QoS requirements of all traffic classes in TSN-based distributed embedded systems. We address each of the three identified challenges for RG_2 to different extents. In the following paragraphs, we describe how we connect the findings of the thesis to each of the challenges towards RG_2.

Firstly, in the scope of RG_2-challenge$_1$, we aimed at automatic network configuration. Our focus in the proposed CP-based offline scheduling approach in [21] was in line with RG_2-challenge$_1$ in the sense that we proposed network constraints and objective functions for automatically generating ST schedules using SMT/OMT solvers. Next in [90], we moved the focus towards applying the automatically generated offline schedules to the TSN networks in simulation environments. We discussed the necessity for the automation and visualization of network configuration. Our experience revealed that even in the case of small networks, manually configuring a large number of activation times for flow instances within our NeSTiNg simulator's simulation framework was excessively laborious, time-consuming, and prone to errors. The proposed framework facilitated offline scheduling, pre-simulation configurations, and interpretation of simulation results in NeSTiNg. We evaluated the applicability of the developed framework for setting up larger networks, as well as an extensive evaluation of use cases was conducted in two of the publications included in this thesis [90, 21]. The automated configuration framework was made available to the research community as an open-source plugin[1] for the NeSTiNg simulator. The proposed framework was proven efficient for the simulation environment under our study, i.e., the NeSTiNg simulator. Due to its modular design, the proposed framework can be further enhanced to support other TSN development platforms.

In the scope of RG_2-challenge$_2$, we further moved the focus towards managing

[1]https://gitlab.com/Scipsybee/automated-tsn-configuration-plugin

the complexity of configuring TSN shaper mechanisms. The porous offline schedules generated by the proposed techniques in [21] allow for using the remaining bandwidth from the ST class for other lower priority classes. In our study, we showed via simulation results that the porous schedules have an effect on the QoS of BE class, i.e., reducing the response times within the network and deadline misses. By benefiting from BE class, we contribute to facilitating the design of systems by reducing the configuration workload for some traffic with less critical timing requirements. We associate this solution, aimed at increasing the QoS of BE, with the next challenge within the RG_2.

The next challenge, RG_2-challenge$_3$, targets reaching an acceptable QoS while using various TSN shaper mechanisms. In line with this challenge, we identified that the bandwidth allocation strategy based on the standard recommendations results in under-reservation of the bandwidth for AVB traffic. As a result, the bandwidth for AVB traffic is limited to the extent that frames in this class miss their deadlines. We addressed this challenge by proposing a technique called BRA to calculate the minimum required bandwidth for configuring the CBS. BRA guarantees the schedulability of all traffic going through the AVB classes. This technique allows the TSN network designers to allocate credits efficiently to AVB traffic classes ensuring the schedulability of the systems. The technique is agnostic of the ST offline schedule, which can be done before this analysis with any scheduling solution that exists in the literature. BRA is based on the existing worst-case RTA, and is pseudo-polynomial.

RG_3: Develop a methodology to extract end-to-end timing information from TSN-based distributed software architectures to support their timing analysis. A challenge towards RG_3 is the end-to-end timing analysis of software architectures of TSN-based distributed embedded systems (RG_3-challenge$_1$). The contributions of this thesis in the scope of RG_3-challenge$_1$ were 1) identification and classification of the sources of the end-to-end timing information in more complex TSN setups; 2) automated methodology to collect the necessary end-to-end timing information and requirements; 3) generating end-to-end timing models to be used as input to end-to-end timing analysis techniques. This set of proposed methodologies is integrated and evaluated as proof of concept within the industrial tool suite RUBUS ICE.

Chapter 5

Conclusions and Future Works

In this chapter, we conclude the thesis and reflect on the remaining research gaps.

5.1 Conclusions

In this thesis, we studied the need for enhancing the usability of the TSN standards for developing predictable distributed embedded systems considering the QoS requirements of all traffic classes in TSN, i.e., ST, AVB, and BE. The TSN standards introduce promising mechanisms for supporting high-bandwidth and low-latency in-vehicle communication. Nevertheless, the adoption of the real-world automotive use cases with TSN remains a challenge.

In Chapter 1, we formulated the overall research goal of the thesis to support the development of TSN-based distributed embedded systems at the stages of system and software modeling, network configuration, and timing analysis while considering the QoS requirements of all traffic classes in TSN. Furthermore, we identified three challenges that need to be addressed to support the aforementioned development stages.

The thesis contributions were introduced in Chapter 3. These contributions target the challenges towards the overall research goal. In particular, we extend the worst-case RTA for TSN to support BE traffic, and subsequently extend the end-to-end data-propagation delay analysis to support synchronized communication of all TSN traffic classes. Moreover, we proposed new schedule synthesis techniques for the ST class in TSN to improve the QoS of the BE class within the network. Then, we investigate automatic configuration techniques to facilitate developing software architectures of TSN-based distributed embedded systems. To this end, we propose a framework to automatically generate TSN network configuration and an end-to-end timing model extraction technique to support the timing analysis of software architectures of these systems. Then, we provided supporting techniques for applying the developed timing analysis techniques at the system and software modeling stage of development. In Chapter 4, we presented an assessment of how each of the thesis contributions can be useful in reaching the research goals of this thesis.

5.2 Future Works

The results presented in this thesis indicated that the proposed techniques are useful to facilitate the system and software modeling, network configuration, and timing analysis stages during the development of TSN-based distributed embedded systems. However, there are remaining gaps to address the challenges towards each research goal that we suggest in this section as the potential Future Works (FW) of this thesis.

FW$_1$: In Section 2.2, we showed that the focus of the state-of-the-art in offline scheduling has moved towards holistic scheduling of the chains of tasks and messages in the TSN-based distributed embedded systems. Such works aim at techniques for simultaneously calculating schedules for a chain of TT tasks and ST messages to optimize the end-to-end data-propagation delays in the chains. The existing techniques face various limitations, e.g., when considering the preemption effects, assigning non-harmonic periods, or task priorities. In the context of this thesis, we suggest investigating the scheduling techniques for TT tasks and ST messages that consider QoS of the lower priority traffic such as BE. Moreover, the contributions of this thesis regarding the timing analysis further enable analysis-based optimization of the end-to-end data-propagation delays.

FW$_2$: In this thesis, we only focused on the challenges for performing timing analysis on the software architectures that are modeled with RCM. However, there are several other software architecture modeling languages and component models that are widely used for developing software architecture for automotive systems, such as AUTOSAR, AMALTHEA, EAST-ADL, AADL. Potential future work is to propose techniques for increasing interoperability and integrating the contributions of this thesis with model-based software development frameworks for distributed embedded systems.

FW$_3$: In this thesis, we have used CP techniques for network scheduling. However, the heuristics scheduling approaches are becoming a popular choice in the research community for solving various scheduling problems in TSN networks as they are easier to be implemented and applied in the industrial tool sets. Besides, the heuristic techniques are faster than CP approach in calculating schedules for scenarios with larger network and traffic. As a future work, we suggest further investigations for heuristic techniques for scheduling ST while considering the QoS of lower priority traffic AVB and BE.

Bibliography

[1] ISO Standard-11898, Road Vehicles–interchange of digital information–controller area network (CAN) for high-speed communication, 1993.

[2] R. Bosch, "CAN with Flexible Data-Rate," *International Organization for Standardization, Geneva, Switzerland, Specification*, vol. 1, p. 2012, 2012.

[3] ISO 17458:2013, "FlexRay Communications System Protocol Specification Version 3.0.1.," 2013.

[4] IEEE 802.1 Time-Sensitive Networking (TSN) Task Group, `https://1.ieee802.org/tsn`.

[5] 802.1Q-2022, "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks," 2022.

[6] L. L. Bello, R. Mariani, S. Mubeen, and S. Saponara, "Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1038–1051, 2019.

[7] M. Ashjaei, L. Lo Bello, M. Daneshtalab, G. Patti, S. Saponara, and S. Mubeen, "Time-Sensitive Networking in Automotive Embedded Systems: State-of-the-art and Research Opportunities," *Journal of Systems Architecture*, vol. 117, p. 102137, 2021.

[8] M. Ashjaei, L. Murselović, and S. Mubeen, "Implications of Various Preemption Configurations in TSN Networks," *IEEE Embedded Systems Letters*, vol. 14, no. 1, pp. 39–42, 2022.

[9] G. Dodig-Crnkovic, "Scientific Methods in Computer Science," in *Proceedings of the Conference for the Promotion of Research in IT at New Universities and at University Colleges in Sweden, Skövde, Suecia*, pp. 126–130, 2002.

[10] J. Jasperneite, P. Neumann, M. Theis, and K. Watson, "Deterministic real-time communication with switched ethernet," in *4th IEEE International Workshop on Factory Communication Systems*, pp. 11–18, 2002.

[11] W. Steiner, G. Bauer, B. Hall, and M. Paulitsch, "Time-Triggered Ethernet," in *Time-Triggered Communication*, pp. 209–248, CRC Press, 2018.

[12] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*, vol. 24. Springer Science & Business Media, 2011.

[13] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A Survey of Scheduling in Time-Sensitive Networking (TSN)," *arXiv preprint arXiv:2211.10954*, 2022.

[14] A. Minaeva and Z. Hanzálek, "Survey on Periodic Scheduling for Time-Triggered Hard Real-Time Systems," *ACM Comput. Surv.*, vol. 54, no. 1, 2021.

[15] V. Gavriluţ, A. Pruski, and M. S. Berger, "Constructive or Optimized: An Overview of Strategies to Design Networks for Time-Critical Applications," *ACM Comput. Surv.*, vol. 55, no. 3, 2022.

[16] W. Steiner, "An Evaluation of SMT-based Schedule Synthesis for Time-triggered Multi-hop Networks," in *2010 31st IEEE Real-time Systems Symposium*, IEEE, 2010.

[17] W. Steiner, "Synthesis of Static Communication Schedules for Mixed-Criticality Systems," in *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, pp. 11–18, 2011.

[18] S. S. Craciunas and R. S. Oliver, "Combined Task-and Network-level Scheduling for Distributed Time-triggered Systems," *Real-Time Systems*, 2016.

[19] S. S. Craciunas and R. S. Oliver, "An Overview of Scheduling Mechanisms for Time-Sensitive Networks," *Proceedings of the Real-Time Summer School*, pp. 1551–3203, 2017.

[20] F. Pozo, G. Rodriguez-Navas, and H. Hansson, "Schedule Reparability: Enhancing Time-triggered Network Recovery Upon Link Failures," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 147–156, 2018.

[21] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, “Synthesising Schedules to Improve QoS of Best-effort Traffic in TSN Networks,” in *Proceedings of the 29th International Conference on Real-Time Networks and Systems*, no. 10, p. 68–77, ACM, 2021.

[22] M. Barzegaran and P. Pop, “Communication Scheduling for Control Performance in TSN-Based Fog Computing Platforms,” *IEEE Access*, vol. 9, pp. 50782–50797, 2021.

[23] D. Bujosa, M. Ashjaei, A. V. Papadopoulos, T. Nolte, and J. Proenza, “HERMES: Heuristic Multi-queue Scheduler for TSN Time-Triggered Traffic with Zero Reception Jitter Capabilities,” in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, p. 70–80, ACM, 2022.

[24] N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, “Window-based Schedule Synthesis for Industrial IEEE 802.1qbv TSN Networks,” in *IEEE International Conference on Factory Communication Systems*, pp. 1–4, 2020.

[25] V. Gavriluţ, L. Zhao, M. L. Raagaard, and P. Pop, “AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN,” *IEEE Access*, vol. 6, pp. 75229–75243, 2018.

[26] Y. Mi, J. Qu, J. Zhang, and M. Yao, “A Scheduling Algorithm of Maximize the Number of Porosity for the Time-Triggered DIMA System,” in *2020 IEEE 3rd International Conference on Electronics Technology*, pp. 68–73, 2020.

[27] A. Berisa, L. Zhao, S. S. Craciunas, M. Ashjaei, S. Mubeen, M. Daneshtalab, and M. Sjödin, “AVB-aware Routing and Scheduling for Critical Traffic in Time-Sensitive Networks with Preemption,” in *Proceedings of the 30th International Conference on Real-Time Networks and Systems*, p. 207–218, ACM, 2022.

[28] O. Hotescu and A. Finzi, “Scheduling Rate Constrained traffic in End Systems of Time-Aware Networks,” in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 1–8, 2021.

[29] M. Vlk, K. Brejchová, Z. Hanzálek, and S. Tang, “Large-Scale Periodic Scheduling in Time-Sensitive Networks,” *Computers Operations Research*, vol. 137, p. 105512, 2022.

[30] M. Pahlevan, N. Tabassam, and R. Obermaisser, “Heuristic List Scheduler for Time Triggered Traffic in Time-Sensitive Networks,” *SIGBED Rev.*, vol. 16, no. 1, p. 15–20, 2019.

[31] M. Pahlevan and R. Obermaisser, “Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks,” in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation*, vol. 1, pp. 337–344, 2018.

[32] H.-J. Kim, K.-C. Lee, M.-H. Kim, and S. Lee, “Optimal Scheduling of Time-Sensitive Networks for Automotive Ethernet Based on Genetic Algorithm,” *Electronics*, vol. 11, no. 6, 2022.

[33] M. Yao, J. Liu, J. Du, D. Yan, Y. Zhang, W. Liu, and A. M.-C. So, “A Unified Flow Scheduling Method for Time-Sensitive Networks,” *Computer Networks*, vol. 233, p. 109847, 2023.

[34] Y. Zheng, S. Wang, S. Yin, B. Wu, and Y. Liu, “Mix-Flow Scheduling for Concurrent Multipath Transmission in Time-Sensitive Networking,” in *2021 IEEE International Conference on Communications Workshops*, pp. 1–6, 2021.

[35] N. Reusch, S. S. Craciunas, and P. Pop, “Dependability-Aware Routing and Scheduling for Time-Sensitive Networking,” *IET Cyber-Physical Systems: Theory & Applications*, vol. 7, no. 3, pp. 124–146, 2022.

[36] X. Zhou, F. He, L. Zhao, and E. Li, “Hybrid Scheduling of Tasks and Messages for TSN-Based Avionics Systems,” *IEEE Transactions on Industrial Informatics*, pp. 1–12, 2023.

[37] L. Lo Bello, M. Ashjaei, G. Patti, and M. Behnam, “Schedulability Analysis of Time-Sensitive Networks with Scheduled Traffic and Preemption Support,” *Journal of Parallel and Distributed Computing*, vol. 144, pp. 153–171, 2020.

[38] M. Ojewale, P. M. Yomsi, and B. Nikolic, “Multi-Level Preemption in TSN: Feasibility and Requirements Analysis,” *2020 IEEE 23rd International Symposium on Real-time Distributed Computing*, pp. 47–55, 2020.

[39] L. Zhao, P. Pop, Z. Zheng, H. Daigmorte, and M. Boyer, “Latency Analysis of Multiple Classes of AVB Traffic in TSN with Standard Credit Behavior using Network Calculus,” *IEEE Transactions on Industrial Electronics*, vol. 68, no. 10, pp. 10291–10302, 2020.

[40] U. D. Bordoloi, A. Aminifar, P. Eles, and Z. Peng, “Schedulability Analysis of Ethernet AVB Switches,” in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*, pp. 1–10, 2014.

[41] M. Ashjaei, G. Patti, M. Behnam, T. Nolte, G. Alderisi, and L. Lo Bello, "Schedulability Analysis of Ethernet Audio Video Bridging Networks with Scheduled Traffic Support," *Real-Time Systems*, vol. 53, pp. 526–577, 2017.

[42] M. A. Ojewale, P. M. Yomsi, and B. Nikolić, "Worst-Case Traversal Time Analysis of TSN with Multi-level Preemption," *Journal of Systems Architecture*, vol. 116, p. 102079, 2021.

[43] M. Ashjaei, G. Patti, M. Behnam, T. Nolte, G. Alderisi, and L. Lo Bello, "Schedulability Analysis of Ethernet Audio Video Bridging Networks with Scheduled Traffic Support," *Real-Time Systems*, 2017.

[44] G. Alderisi, G. Patti, and L. L. Bello, "Introducing Support for Scheduled Traffic over IEEE Audio Video Bridging Networks," in *2013 IEEE 18th Conference on Emerging Technologies Factory Automation*, pp. 1–9, 2013.

[45] J. A. R. De Azua and M. Boyer, "Complete Modelling of AVB in Network Calculus Framework," in *Proceedings of the International Conference on Real-Time Networks and Systems*, p. 55–64, ACM, 2014.

[46] M. Boyer, H. Daigmorte, N. Navet, and J. Migge, "Performance Impact of the Interactions between Time-Triggered and Rate-Constrained Transmissions in TTEthernet," in *8th European Congress on Embedded Real Time Software and Systems* , 2016.

[47] L. Zhao, P. Pop, 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*, pp. 25–36, 2018.

[48] L. Zhao, P. Pop, and S. S. Craciunas, "Worst-Case Latency Analysis for IEEE 802.1Qbv Time Sensitive Networks Using Network Calculus," *IEEE Access*, vol. 6, pp. 41803–41815, 2018.

[49] L. Maile, K.-S. Hielscher, and R. German, "Network Calculus Results for TSN: An Introduction," in *2020 Information Communication Technologies Conference*, pp. 131–140, 2020.

[50] L. Zhao, Y. Yan, and X. Zhou, "Minimum Bandwidth Reservation for CBS in TSN With Real-Time QoS Guarantees," *IEEE Transactions on Industrial Informatics*, pp. 1–12, 2023.

[51] T. L. Mai, N. Navet, and J. Migge, “A Hybrid Machine Learning and Schedulability Analysis Method for the Verification of TSN Networks,” in *2019 15th IEEE International Workshop on Factory Communication Systems*, pp. 1–8, 2019.

[52] T. L. Mai, N. Navet, and J. Migge, “On the use of Supervised Machine Learning for Assessing Schedulability: Application to Ethernet TSN,” in *Proceedings of the 27th International Conference on Real-Time Networks and Systems*, p. 143–153, ACM, 2019.

[53] T. Long Mai and N. Navet, “Improvements to Deep-Learning-based Feasibility Prediction of Switched Ethernet Network Configurations,” in *Proceedings of the 29th International Conference on Real-Time Networks and Systems*, p. 89–99, ACM, 2021.

[54] J. Cao, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien, “Tight Worst-case Response-time Analysis for Ethernet AVB using Eligible Intervals,” in *2016 IEEE World Conference on Factory Communication Systems*, pp. 1–8, 2016.

[55] D. Maxim and Y.-Q. Song, “Delay Analysis of AVB Traffic in Time-Sensitive Networks (TSN),” in *Proceedings of the 25th International Conference on Real-Time Networks and Systems*, p. 18–27, ACM, 2017.

[56] R. J. Bril, “Independent WCRT Analysis for the Best-Effort Class BE in Ethernet AVB,” in *2022 IEEE 18th International Conference on Factory Communication Systems*, pp. 1–4, 2022.

[57] J. Cao, M. Ashjaei, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien, “An Independent yet Efficient Analysis of Bandwidth Reservation for Credit-based Shaping,” in *2018 14th IEEE International Workshop on Factory Communication Systems*, pp. 1–10, 2018.

[58] R. J. Bril, H. Hassani, P. J. Cuijpers, and G. Nelissen, “Cost of Robustness of Independent WCRT Analysis for CBS of Ethernet AVB Using Eligible Intervals,” in *2023 IEEE 19th International Conference on Factory Communication Systems (WFCS)*, pp. 1–4, 2023.

[59] AUTOSAR Consortium, AUTOSAR Techincal Overview [online], Release 4.1, Rev.2, Ver.1.1.0., `http://autosar.org`.

[60] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K.-L. Lundbäck, “Supporting Timing Analysis of Vehicular Embedded Systems through the Refinement of Timing Constraints,” *Software & Systems Modeling*, vol. 18, no. 1, pp. 39–69, 2019.

[61] L. Zhao, F. He, E. Li, and J. Lu, "Comparison of Time-Sensitive Networking (TSN) and TTEthernet," in *2018 IEEE/AIAA 37th Digital Avionics Systems Conference*, pp. 1–7, 2018.

[62] M. Günzel, K.-H. Chen, N. Ueter, G. v. d. Brüggen, M. Dürr, and J.-J. Chen, "Timing Analysis of Asynchronized Distributed Cause-Effect Chains," in *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium*, pp. 40–52, 2021.

[63] S. Mubeen, J. Mäki-Turja, and M. Sjödin, "Support for End-to-end Response-time and Delay Analysis in the Industrial Tool Suite: Issues, Experiences and a Case Study," *Computer Science and Information Systems*, vol. 10, no. 1, pp. 453–482, 2013.

[64] M. Ashjaei, N. Khalilzad, S. Mubeen, M. Behnam, I. Sander, L. Almeida, and T. Nolte, "Designing End-to-end Resource Reservations in Predictable Distributed Embedded Systems," *Real-Time Systems*, vol. 53, pp. 916–956, 2017.

[65] J. Schlatow, M. Mostl, S. Tobuschat, T. Ishigooka, and R. Ernst, "Data-Age Analysis and Optimisation for Cause-Effect Chains in Automotive Control Systems," in *2018 IEEE 13th International Symposium on Industrial Embedded Systems*, pp. 1–9, 2018.

[66] J. Martinez, I. Sañudo, and M. Bertogna, "Analytical Characterization of End-to-end Communication Delays with Logical Execution Time," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2244–2254, 2018.

[67] T. Klaus, F. Franzmann, M. Becker, and P. Ulbrich, "Data Propagation Delay Constraints in Multi-Rate Systems: Deadlines vs. Job-Level Dependencies," in *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, p. 93–103, ACM, 2018.

[68] R. Bi, X. Liu, J. Ren, P. Wang, H. Lv, and G. Tan, "Efficient Maximum Data Age Analysis for Cause-Effect Chains in Automotive Systems," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, p. 1243–1248, ACM, 2022.

[69] 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 *IEEE Real-Time Systems Symposium*, IEEE Communications Society, 2009.

[70] M. Dürr, G. V. D. Brüggen, K.-H. Chen, and J.-J. Chen, “End-to-End Timing Analysis of Sporadic Cause-Effect Chains in Distributed Systems,” *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 5, 2019.

[71] K.-B. Gemlau, L. KÖHLER, R. Ernst, and S. Quinton, “System-level Logical Execution Time: Augmenting the Logical Execution Time Paradigm for Distributed Real-time Automotive Software,” *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 2, 2021.

[72] T. Kloda, A. Bertout, and Y. Sorel, “Latency Analysis for Data Chains of Real-Time Periodic Tasks,” in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation*, vol. 1, pp. 360–367, 2018.

[73] L. Köhler, P. Hertha, M. Beckert, A. Bendrick, and R. Ernst, “Robust Cause-Effect Chains with Bounded Execution Time and System-Level Logical Execution Time,” *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 3, 2023.

[74] H. Teper, M. Günzel, N. Ueter, G. von der Brüggen, and J.-J. Chen, “End-To-End Timing Analysis in ROS2,” in *2022 IEEE Real-Time Systems Symposium*, pp. 53–65, 2022.

[75] M. Ashjaei, S. Mubeen, J. Lundbäck, M. Gålnander, K.-L. Lundbäck, and T. Nolte, “Modeling and timing analysis of vehicle functions distributed over switched ethernet,” in *43rd Annual Conference of the IEEE Industrial Electronics Society*, pp. 8419–8424, 2017.

[76] S. Mubeen, M. Gålnander, J. Lundbäck, and K.-L. Lundbäck, “Extracting Timing Models from Component-based Multi-Criticality Vehicular Embedded Systems,” in *Information Technology-New Generations: 15th International Conference on Information Technology*, pp. 709–718, Springer, 2018.

[77] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, “System level performance analysis–the SymTA/S approach,” *IEE Proceedings-Computers and Digital Techniques*, vol. 152, no. 2, pp. 148–166, 2005.

[78] A. Arestova, M. Martin, K.-S. J. Hielscher, and R. German, “A service-oriented Real-Time Communication Scheme for AUTOSAR Adaptive using OPC UA and Time-Sensitive Networking,” *Sensors*, vol. 21, no. 7, p. 2337, 2021.

[79] D. Bruckner, M.-P. Stănică, R. Blair, S. Schriegel, S. Kehrer, M. Seewald, and T. Sauter, “An Introduction to OPC UA TSN for Industrial Communication Systems,” *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1121–1131, 2019.

[80] A. Bucaioni, P. Pelliccione, and S. Mubeen, “Modelling Centralised Automotive E/E Software Architectures,” *Advanced Engineering Informatics*, vol. 59, p. 102289, 2024.

[81] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, and K.-L. Lundback, “The Rubus Component Model for Resource-Constrained Real-Time Systems,” in *2008 International Symposium on Industrial Embedded Systems*, pp. 177–183, 2008.

[82] S. Mubeen, H. B. Lawson, J. Lundbäck, M. Gålnander, and K.-L. Lundbäck, “Provisioning of Predictable Embedded Software in the Vehicle Industry: The Rubus Approach,” in *4th International Workshop on Software Engineering Research and Industry Practice, located at the 39th International Conference on Software Engineering*, pp. 3–9, 2017.

[83] A. Bucaioni, L. Addazi, A. Cicchetti, F. Ciccozzi, R. Eramo, S. Mubeen, and M. Sjödin, “MoVES: A Model-Driven Methodology for Vehicular Embedded Systems,” *IEEE Access*, vol. 6, pp. 6424–6445, 2018.

[84] A. Bucaioni, M. Becker, J. Lundbäck, and H. Mackamul, “From AMALTHEA to RCM and Back: a Practical Architectural Mapping Scheme,” in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 537–544, 2020.

[85] A. Bucaioni, V. Dimic, M. Gålnander, H. Lönn, and J. Lundbäck, “Transferring a Model-Based Development methodology to the automotive industry,” in *2021 22nd IEEE International Conference on Industrial Technology*, vol. 1, pp. 762–767, 2021.

[86] J. Jiang, Y. Li, S. H. Hong, A. Xu, and K. Wang, “A Time-Sensitive Networking (TSN) Simulation Model Based on OMNET++,” in *2018 IEEE International Conference on Mechatronics and Automation*, pp. 643–648, 2018.

[87] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel, “NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++,” in *IEEE Proceedings of the International Conference on Networked Systems*, 2019.

[88] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, “End-to-end Timing Analysis of Cause-effect Chains in Automotive Embedded Systems,” *Journal of Systems Architecture*, vol. 80, pp. 104–113, 2017.

[89] D. Hellmanns and J. Falk, “NeSTiNg - Network Simulator for Time-Sensitive Networking [Online, 2020-10-26].”

[90] B. Houtan, A. Bergström, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, “An Automated Configuration Framework for TSN Networks,” in *2021 22nd IEEE International Conference on Industrial Technology*, vol. 1, pp. 771–778, 2021.

[91] B. Houtan, M. O. Aybek, M. Ashjaei, M. Daneshtalab, M. Sjödin, J. Lundbäck, and S. Mubeen, “End-to-end Timing Modeling and Analysis of TSN in Component-Based Vehicular Software,” in *2023 IEEE 26th International Symposium on Real-Time Distributed Computing*, pp. 126–135, 2023.

[92] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Afshar, and S. Mubeen, “Schedulability Analysis of Best-Effort Traffic in TSN Networks,” in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 1–8, 2021.

[93] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, “Supporting End-to-end Data Propagation Delay Analysis for TSN-Based Distributed Vehicular Embedded Systems,” *Journal of Systems Architecture*, vol. 141, p. 102911, 2023.

[94] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, “Bandwidth Reservation Analysis for Schedulability of AVB Traffic in TSN,” in *the 25th IEEE International Conference on Industrial Technology*.

Part II

Included Papers

Chapter 6

Paper A: An Automated Configuration Framework for TSN Networks

Bahar Houtan, Albert Bergström, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Saad Mubeen.

In the 22^{nd} IEEE International Conference on Industrial Technology (ICIT), 2021.

Abstract

Designing and simulating large networks, based on the Time Sensitive Networking (TSN) standards, require complex and demanding configuration at the design and pre-simulation phases. Existing configuration and simulation frameworks support only manual configuration of TSN networks. This hampers the applicability of these frameworks to large-sized TSN networks, especially in complex industrial embedded system applications. This paper proposes a modular framework to automate offline scheduling in TSN networks to facilitate the design-time and pre-simulation automated network configurations as well as interpretation of the simulations. To demonstrate and evaluate the applicability of the proposed framework, a large TSN network is automatically configured and its performance is evaluated by measuring end-to-end delays of time-critical flows in a state-of-the-art simulation framework, namely NeSTiNg.

6.1 Introduction

Time-Sensitive Networking (TSN) is a set of standards, developed by the IEEE 802.1 TSN task group [1], to support high-bandwidth, time-critical, and low-latency communication over the switched Ethernet. These standards offer several novel features, e.g., a common notion of time through clock synchronization, resource reservation for various types of traffic, traffic shaping, scheduled traffic support, frame preemption, and much more. There is a huge interest in utilizing TSN in time-critical applications, in particular in the automotive [2] and industrial automation domains [3]. However, there are several challenges that are encountered when utilizing TSN in industrial applications. One core challenge is to perform the TSN network configuration during the design, analysis and simulation phases, while taking the applications' timing requirements into account. These applications often require complex configuration measures at the design and pre-simulation phases, such as creating offline schedules for the scheduled traffic. The existing network design and simulation frameworks for TSN lack automation in the network configuration. Manually configuring a large number of parameters associated to various types of traffic in TSN can be time consuming, error prone and cumbersome for the network designers.

To address this challenge, this paper proposes an automated framework that facilitates the design-time and pre-simulation network configuration of TSN networks. The proposed framework uses one of the state-of-the-art open-source TSN simulation platforms, namely NeSTiNg [4, 5]. The source code for the proposed framework is openly provided in gitlab[1]. The proposed framework is desinged in a modular way and it uses the Extensible Markup Language (XML) format for the information exchange, which allow the framework to be easily adapted to other simulation platforms. The main contributions in the proposed framework are as follows:

- integrating the traffic generation and optimized network schedule synthesis to the NeSTiNg open-source simulation framework for TSN;
- addressing the flow configuration complexity by automatically translating configuration for the scheduled flows' into the NeSTiNg syntax compliant configuration;
- addressing the gate states configuration complexity by automatically translating the synthesized schedules into the NeSTiNg syntax compliant configuration; and
- finally, allowing to automatically configure the configuration files by a Graphical User Interface (GUI) that is integrated into the simulation framework.

[1]https://gitlab.com/Scipsybee/automated-tsn-configuration-plugin

6.2 Background and Related Work

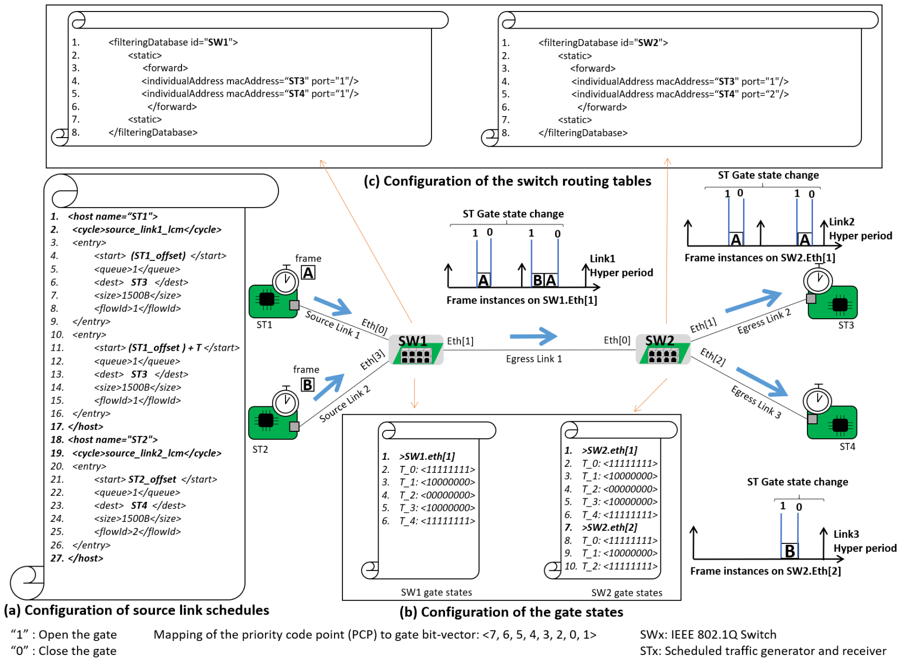
The Time-Aware Shaper (TAS) mechanism in the IEEE 802.1Qbv standard allows arbitration of various types of traffic based on priorities at a TSN switch's egress port. TSN composes eight traffic priorities that are encoded by a 3-bit priority code point (PCP), which is associated with the index of 8 priority queues at TSN switch's egress port. For example, to reserve the switch bandwidth for the highest priority real-time class, the Scheduled Traffic (ST) with $PCP = 7$, a time-division multiple access transmission selection algorithm is used. Egress data flow from each priority queue is controlled by a gate. Based on this algorithm, the offline schedules must be time-stamped in a Gate Control List (GCL), which contains the starting and finishing times of allowed time slots for each queue and associated configuration of the gate states (GS).

There are several works that discuss automatic synthesis of optimized gate schedules in TSN by using solvers like the Satisfiability/Optimized Modulo Theorem (S/OMT), e.g., the works by Craciunas et al. [6], Hashemi et al. [7], Schneider and Santos et al. [8], and Gavrilut et al. [9]. The work by Pahlevan et al [10] uses the Genetic Algorithm to automatically synthesize the gate schedules. There are several simulation frameworks for pre-implementation evaluation of TSN networks. OMNeT++ is an open-source discrete event simulation platform, primarily used for constructing simulations of networks[2]. INET[3] is another widely used open-source simulation framework that models wired, wireless and mobile networks. Core4INET [11] is an open-source TSN simulation framework that is built on OMNeT++. Core4INET allows simulation of TSN networks based on various standards, e.g., IEEE 802.1Q, IEEE P802.1p VLANs and Priorities, IEEE 802.1 AVB and TTEthernet (AS6802). Another notable example of the TSN simulation frameworks is NeSTiNg [4]. NeSTiNg supports several TSN features, including the scheduled traffic (IEEE 802.1Qbv), frame preemption (IEEE Std 802.1Qbu and IEEE Std 802.3br), credit-based shaper (IEEE Std 802.1Qav), time synchronization (IEEE Std 802.1AS). Schedule and routing in NesTiNg can be done through insertion of XML files, with a strict syntax. This feature allows NesTiNg to be seamlessly integrated to the proposed framework. Hence, the network simulations with NesTiNg can be facilitated by automatic generation of XML configuration files for flow schedules, flow attributes and gate states. Both Core4INET and NeSTiNg require significant amount of manual configurations at the pre-simulation phase as well as for interpretation of the simulation results. The framework proposed in this paper augments automation in these frameworks by means of an automated configuration framework for TSN.

[2]https://omnetpp.org/

[3]https://inet.omnetpp.org/

6.3 Automated Configuration Framework



```
<filteringDatabase id="SW1">
    <static>
        <forward>
        <individualAddress macAddress="ST3" port="1"/>
        <individualAddress macAddress="ST4" port="1"/>
          </forward>
    <static>
</filteringDatabase>
```

```
<filteringDatabase id="SW2">
    <static>
        <forward>
        <individualAddress macAddress="ST3" port="1"/>
        <individualAddress macAddress="ST4" port="2"/>
          </forward>
    <static>
</filteringDatabase>
```

```
<host name="ST1">
 <cycle>source_link1_lcm</cycle>
 <entry>
        <start> (ST1_offset) </start>
        <queue>1</queue>
        <dest>  ST3  </dest>
        <size>1500B</size>
        <flowId>1</flowId>
 </entry>
 <entry>
        <start> (ST1_offset ) + T </start>
        <queue>1</queue>
        <dest>  ST3  </dest>
        <size>1500B</size>
        <flowId>1</flowId>
 </entry>
</host>
<host name="ST2">
 <cycle>source_link2_lcm</cycle>
 <entry>
        <start> ST2_offset  </start>
        <queue>1</queue>
        <dest>  ST4 </dest>
        <size>1500B</size>
        <flowId>2</flowId>
 </entry>
</host>
```

```
>SW1.eth[1]
T_0: <11111111>
T_1: <10000000>
T_2: <00000000>
T_3: <10000000>
T_4: <11111111>
```

```
>SW2.eth[1]
T_0: <11111111>
T_1: <10000000>
T_2: <00000000>
T_3: <10000000>
T_4: <11111111>
>SW2.eth[2]
T_0: <11111111>
T_1: <10000000>
T_2: <11111111>
```

Figure 6.1. Various sources of the configuration complexity.

6.3.1 Configuration Complexity in TSN

This subsection demonstrates the level of complexity in the configuration of TSN networks, in particular, the configuration that is required to simulate TSN networks using the NeSTiNg framework. We demonstrate various configuration parameters by an example shown in Figure 6.1. The example illustrates a TSN network consisting of two transmitter end stations, **ST1** and **ST2**, and two receiver end stations, **ST3** and **ST4**. The end stations are connected via two TSN switches, **SW1** and **SW2**. **ST1** transmits a periodic flow to **ST3**. Whereas, **ST2** sends a periodic flow to **ST4**. The period of **ST2**'s flow is double the period of **ST1**'s flow.

The network configuration is performed in three steps: (i) configuration of offline schedules for the scheduled traffic on each link, (ii) configuration of gate states in egress queues of each TSN switch port, and (iii) configuration of switch routing tables.

Configuration of the Source Link Schedules

In the example shown in Figure 6.1, source link schedules are specified by the text in the box (a). In general, there are 27 lines in the flow configuration file. Subsequently, 14 and 7 lines of code are needed to indicate the traffic characteristics of the data flows transmitted from **ST1** and **ST2**. One drawback of the current configuration method is that the traffic settings should be assigned based on the flow generating source end stations. This method becomes challenging when a source end station needs to transmit scheduled traffic with different periods, different data size of flows and via different queues to different destination end stations.

The NeSTiNg configuration approach does not support convenient modelling for TSN flows. In the current modelling perspective, traffic flows must be specified based on the flow generating source end station. Hence, the flow schedules from each source end station should be specified based on the flow offsets in chronological order in an XML file. The bottleneck in this method is the extra effort for the network designers at the traffic configuration stage. Because in case of having different flow profiles from a source end station, it is necessary to add each periodic instance of every flow[4] until *hyper period* of the flows as a new schedule entry to the XML configuration file. The hyper period of the flows is the least common multiple of periods of all flows. It is extremely laborious, time-consuming and error-prone to perform manual configuration of a large number of activation times of the flow instances, even in the case of small networks.

Network Gate State Configuration

In order to guarantee timely transmission of the ST flows in the network, the gate opening and closing times in the egress ports of each TSN switch must be assigned, while taking the timing requirements on the flows into account. The TAS provides a mechanism of opening and closing gates of the egress queues, which creates reserved time slots on the egress link. These time slots assign the link capacity to the flows, which are scheduled to pass through the egress link. There are 8 traffic classes that correspond to a gate in the TAS. In order to calculate the reserved time slots on the link, the trace of all the frames passing through the link must be arranged and defined as gate states at the egress port of the TSN switch connected to that link. To specify the ST frame transmission slots on the link, it is necessary to calculate the offset of the frames until the hyper period of the crossing flows from the link. Assigning the gate

[4]A flow instance is represented by a frame

states can become very complex due to increase in the number of slots that need to be reserved on the link. Conventionally, the gate states can be assigned by manually drawing the network trace and inspecting the arrival times of the ST flows on each link. The box (b) in Figure 6.1 shows a sample trace of the two flows on egress link 1 from **SW1** and egress links 2 and 3 from **SW2**. The gate states in the example shown in Figure 6.1 can be manually configured by drawing the traces of the network. However, manually performing the gate-states configurations for large-sized networks that include a large number of flows of different traffic classes is impractical, time-consuming and error prone. Even if the schedules for the flows are calculated using optimized schedule generation engines, manual translation of the generated schedules to comply with the input configuration syntax for the simulation frameworks (like NeSTiNg) requires a massive pre-simulation effort.

Switch Routing Table Configuration

A routing table statically defines the paths to forward traffic from the source end station to the destination end station's address. The box (c) in Figure 6.1 shows the routing table configurations. In case of larger networks, there might be multiple paths to a destination end station, which calls for configuring the routing table of each switch. This requires the designer to acquire knowledge of all the connections between switches and switch port numbers connected to each end station.

It can be seen from the above discussion that simulation of TSN networks requires complex pre-simulation configurations of various parameters. In the case of large TSN networks, manually setting up these configurations becomes time consuming, error prone and impractical. Therefore, an automatic configuration framework is essential, in particular for designing, simulating and evaluating large industrial network.

6.3.2 Modular Architecture of the Proposed Framework

This subsection presents the modular architecture of the proposed automatic configuration framework as depicted in Figure 6.2. The proposed framework consists of four modules: (i) traffic generator, (ii) schedule synthesizer, (iii) automated configurator, and (iv) output and results interpreter.

Traffic Generator

The traffic generator module generates attributes of the TSN flows that are required for traffic configuration in the simulator. This module provides flow-based traffic configuration input to the simulator. In this module, periodic flows (a.k.a. streams) are

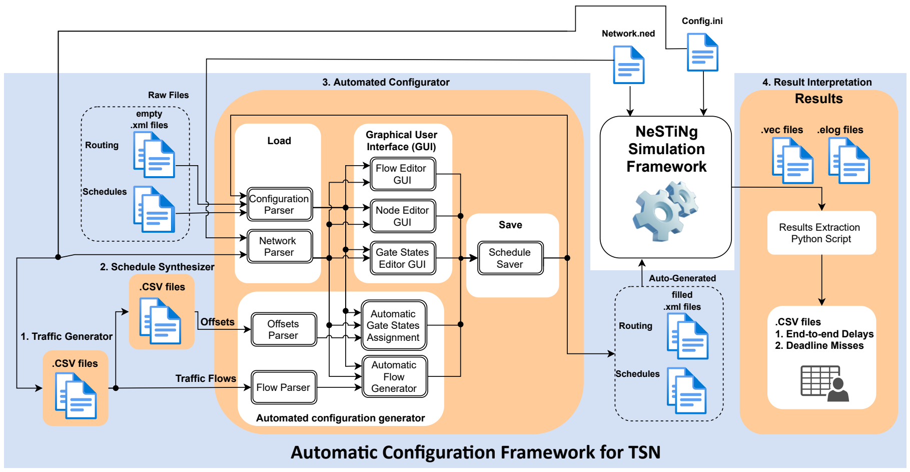


Figure 6.2. Modular Design of the proposed automated configuration framework.

represented by a tuple, S. Each flow s_i, in the set of flows, is represented by Eq. (6.1):

$$S := \langle \{s_1, \ldots, s_{|S|}\}, \forall [V_a, V_b] \in L : hp_{[V_a,V_b]} \rangle \tag{6.1}$$

where, V_a and V_b specify a pair of end stations/switches connected via a link denoted by $[V_a, V_b]$, which is a member of the network link set, denoted by L. The parameter, $hp_{[V_a,V_b]}$, represents the hyper period of the set of flows, which is the least common multiple of periods of flows crossing the link $[V_a, V_b]$. Furthermore, the flow profiles are characterized by Eq. (6.2):

$$s_i := \langle src_i, t_i, d_i, l_i, p_i, r_i, dest_i \rangle \tag{6.2}$$

where, i is the unique ID of each flow. The transmitter end station's ID is represented by src_i. The source end station transmits flows of data to the destination end station with the ID $dest_i$. The period and deadline of the flow are represented by t_i and d_i, respectively. The parameter l_i stores the transmission duration of the flow. The flow priority is denoted by p_i. Finally, r_i holds the list of switches in the path from the source end station to the destination end station.

In addition to setting the flow attributes, this module enables editing of the link attributes, which affect the transmission duration of frames and synchronization of the transmitter and receiver end stations. Eq. (6.3) specifies the link attributes.

$$\forall [V_a, V_b] \in L := \langle speed, delay, tick, queue \rangle \tag{6.3}$$

The parameter, $speed$, indicates the maximum allowed bandwidth on the link in bit/s. The propagation delay on the link is denoted by $delay$. $tick$ is the synchronization factor of the physical layer, according to IEEE 802.1AS. Finally, the maximum number of queues is represented by $queue$. The traffic generator module receives a graph topology of the network. The outputs of this module are flow profiles that are stored in a Comma Separated Vector (CSV) file to be used by the other framework modules.

Schedule Synthesizer

The schedule synthesizer is a constraint-based solver that is intended to find solution of a scheduling optimization objective function and a set of real-time constraints on the ST flows. The schedule synthesis module produces offsets that ensure the timing requirements of the time-triggered ST traffic at each hop in the flow's path are satisfied. We specifically employ an open-source S/OMT solver, namely Z3[5] that features both satisfiability and optimization solvers. These solvers are automated theorem provers that search for solutions by examining the satisfiability of possible combination of solutions in a search tree. This module receives a CSV file containing the standard flow set and link set specifications presented by Eq. (6.1), Eq. (6.2) and Eq. (6.3). The schedule synthesis module specifies transmission time of each ST flow on each link. In this work, we apply the network modelling approach presented in [6]. Where, frame i in the k^{th} flow is indicated by $f_{i,k}$. The path from a transmitter end station, V_a, to a receiver end station, V_b, is shown as the set, $[[V_a, V_1],\ [V_1, V_2], ..., [V_i, V_j], ..., [V_{n-2}, V_{n-1}], [V_n, V_b]]$. The transmission gate schedules on each link must be defined by a set of well-defined tuples to be readable by the automatic configuration generation module. Therefore, we consider the following attributes for each frame $f_{i,k}^{[V_i,V_j]}$, $<\phi,\ Len>$. Where, ϕ represents the transmission time, and Len indicates the allowed margin on the link for complete transmissions of the frame. As a results, the list of gate transaction time stamps of each TSN switch is shown in Eq. (6.4):

$$\begin{aligned} \forall [V_a, V_b] \in L, \forall s_{\{1,...,|S|\}} \in S, \forall f_{i,k}^{[V_a,V_b]} \in s_i^{[V_a,V_b]} : \\ [f_{i,k}^{[V_a,V_b]}.\phi, f_{i,k}^{[V_a,V_b]}.\phi + f_{i,k}^{[V_a,V_b]}.Len] \end{aligned} \tag{6.4}$$

where, the list of gate states at a switch's egress port is defined by the set of allowed time margins to transmit frames on the associated egress link. The start and length of the time margins are denoted by the flow's frame offset and a predefined window size respectively.

[5]https://pypi.org/project/z3-solver/

Automated Configurator

The automated configuration module is realized as a plug-in for the OMNET++ TSN simulation framework. Hence, it is easliy integrable with the NeSTiNg simulation framework. The plug-in consists of two main sub-modules.

- Graphical User Interface (GUI): Through the GUI sub-module, the flows, schedules, routing settings and gate states can be created and modified manually.
- Automated Configuration Generator: This sub-module receives the simulation setting files from the traffic generation and schedule synthesis modules. The settings are then translated into the NeSTiNg configuration files.

The NeSTiNg XML schedule and routing configuration files can be edited both by the GUI and Automated Configuration Generator. If the XML files contain flow entries, they can be parsed by the load sub-module, which can be further edited and visualized in the GUI. Also, new entries can be written on the files through the GUI or the Automated Configuration Generator. The save sub-module translates the received configurations into the NeSTiNg compliant configurations.

Output and Results Interpreter

While it is possible to collect certain useful statistics (e.g., *end-to-end delays*) in the NeSTiNg framework, the manual inspection of numerous recorded parameters for each component in large networks is impractical. The end-to-end delay is a time interval from the transmission of a frame from its source end station until it has arrived at the destination end station. Besides, OMNeT++ and NeSTiNg simulation records are frame-based as they use source-based traffic configuration. Therefore, these simulation frameworks lack a built-in functionality for extraction of the TSN flow metrics. Python scripts are suggested by OMNeT++ official technical guide in [12] to retrieve insights from the simulator's vector (VEC) files. In order to acquire end-to-end delays of the frames, a python script is devised, which constitutes the final sub-module of the framework as shown in Figure 6.2. The script traces the vector files, records the transmission and reception times of each flow, and calculates the flow's end-to-end delay. In addition to the aforementioned python script, the final module in the proposed framework also features an OMNeT++ integrated sub-module to extract the maximum end-to-end delays of the flows. The flow extraction component of the plug-in also takes the VEC files containing the transmission and reception times of the flow as input. Thereafter, a CSV file is created with the maximum end-to-end delays corresponding to each TSN flow.

6.4 Implementation

This section presents a proof-of-concept implementation of the proposed framework. As shown in Figure 6.3, the automated configuration module performs configurations both in manual and automatic mode. There are several configuration algorithms that can be utilized to obtain an optimum network timing behaviour. The proposed framework is implemented as a TSN plugin, which is integrated to the NeSTiNg simulator using the OMNeT++ Java application programming interface and the plug-in development environment in OMNeT++.

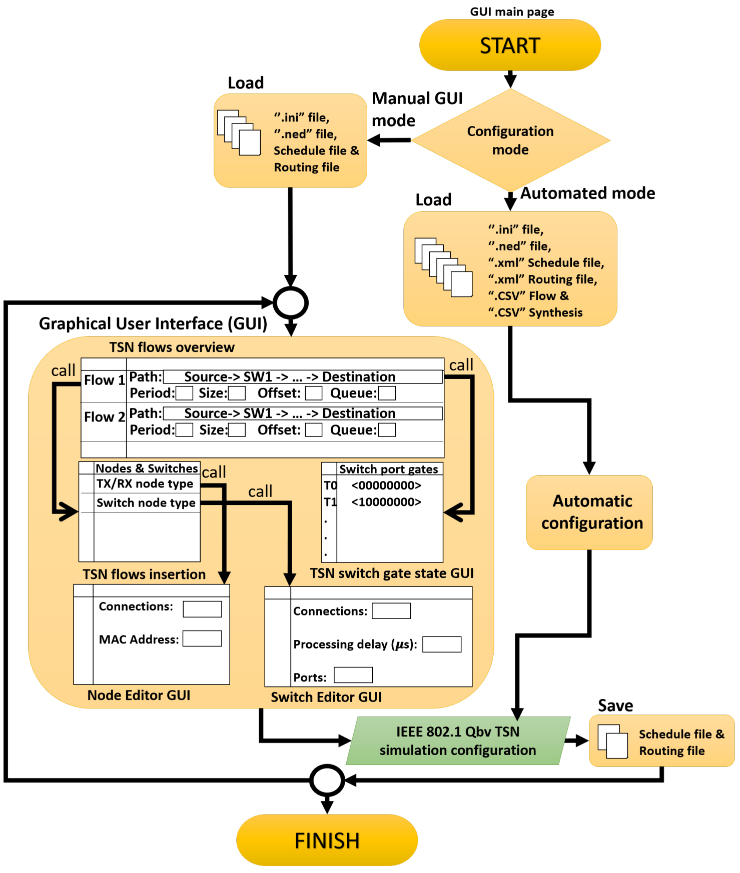

GUI main page
START
Configuration mode
Manual GUI mode
Load
".ini" file, ".ned" file, Schedule file & Routing file
Automated mode
Load
".ini" file, ".ned" file, ".xml" Schedule file, ".xml" Routing file, ".CSV" Flow & ".CSV" Synthesis
Graphical User Interface (GUI)
TSN flows overview
call
Flow 1 Path: Source-> SW1 -> ... -> Destination
Period: Size: Offset: Queue:
Flow 2 Path: Source-> SW1 -> ... -> Destination
Period: Size: Offset: Queue:
call
Nodes & Switches
TX/RX node type
Switch node type
call
call
Switch port gates
T0 <00000000>
T1 <10000000>
TSN flows insertion
TSN switch gate state GUI
Connections:
MAC Address:
Node Editor GUI
Connections:
Processing delay (μs):
Ports:
Switch Editor GUI
Automatic configuration
IEEE 802.1 Qbv TSN simulation configuration
Save
Schedule file & Routing file
FINISH

Figure 6.3. Automated flow and gate configurator flowchart.

6.4.1 Framework Configuration Files

Pre-simulation Configuration

There is a set of files to be configured before starting the simulation process. The automatic configuration of the pre-simulation process is performed by configuring the following model files.

- Simulation initialization (INI) file: this file contains simulator specific configuration options. This file can be either edited directly or modified in a wizard that is embedded in the OMNeT++ IDE [13]. Our framework uses this file to obtain the Media Access Control (MAC) addresses of the end stations and processing delays in TSN switches.
- Network Topology Description (NED) file: the component modules, sub-modules, channels and overloaded component types can be defined in this file. The OMNeT++ IDE allows setting this file in visual or non-visual modes [13]. Our framework reads the NED files and obtains information such as network topology and available paths from the source end station to the destination end station.
- XML schedule file: transmission schedules for ST flows must be defined and inserted in the NeSTiNg simulator using XML files. Our proposed framework enables automated schedule generation, editing of schedules by a GUI and visualization of each frame in a flow by accessing this file.
- XML routing file: this file statically defines the paths to forward traffic according to the destination end station's address. The XML routing file defines the forwarding database for each switch. By using information from the NED file, such as topology and available paths from the source end station to the destination end station, the framework specifies fixed routes, thereby generating the XML routing file.
- CSV flow file: the set of data received from the traffic generator module is saved in this file. The file contains flows' profiles such as queue index, frame size and destination end station, which are used in the automated mode to further configure the ST flows in the XML schedule file.
- CSV gate synthesis file: this file contains the set of data received from the schedule synthesizer, including offset of frames on each egress link. Accordingly, this file can be used in the automated mode to set the switch gate states in the XML schedule file.

Post-simulation Files

The NeSTiNg simulator generates vector outputs of the network event parameters and their occurrence times after the simulation. These files include:

- Results' vectors (VEC): after the simulation, the events and time stamps are recorded in the vector files. The proposed framework's post-simulation analysis module uses these files to obtain end-to-end delay and deadline miss metrics.
- Event logs file (elog): This file stores the frame transmission traces that can be visualized in OMNeT++ by the GUI. We refer the reader to [5] for further details about the elog files.
- End-to-end delay and deadline miss log (CSV): this file contains interpreted results using the simulator's VEC files. This file includes the end-to-end delays and the number of deadline misses experienced by the flows.

6.4.2 Configuration of Inputs and GUI Layout

The automatic configuration process supports two modes: (i) manual; and (ii) automatic. Figure 6.3 demonstrates the flowchart of the manual and automated configuration input methods. In the manual mode, the GUI allows the user to select input files from the work space. The manual mode requires the NeSTiNg's INI and NED files, as well as two empty XML files to be loaded in the fields; "INI file", "NED file", "XML schedule file" and "XML routing file". Manual entries for schedule configurations are saved in the associated XML file. The routing database is saved in the XML routing file based on existing physical channels, as described in the NED file. The GUI allows to append the flow path entries through the "TSN flows insertion" function, which enables to select the source and destination end stations, as well as the specification of static route among the switches within the path from the source end station to the destination end station. After specifying the flow path, other flow attributes can be viewed and adjusted in the "TSN flows overview" function. These attributes include offset, flow size and a dedicated switch queue for the flow. Furthermore, physical characteristics of end stations and switches can be adjusted by calling the "Node Editor GUI" and "Switch Editor GUI" functions. The "TSN switch gate state GUI" function enables inserting the gate states for each port in a TSN switch. Besides, it assists to adjust and modify the gate configurations by visualizing the gate states specified for a frame scheduled on each link. The results of the manual GUI are finally saved in the XML files that are used by the simulation framework.

In the automated configuration mode, insertion of the flow and gate-state configurations can be performed automatically by loading well-defined flow entry and gate-state data files, which must be provided by ".CSV flows" and ".CSV Synthesis" fields in the main GUI along with the rest of the configuration files. We note here that the automated configuration data are prepared by preceding automated sub-modules: traffic generator and schedule synthesizer. Therefore, the automated mode removes the risk of human errors that can be introduced with manual configurations. After the automatic generation of the schedules and routing configurations, the output can be visualized, inspected and adjusted by the manual GUI. The save sub-module translates configurations to the NeSTiNg compatible syntax. We refer the reader to [14] for further details about the implementation and user manual of the GUI. If the proposed framework is to be integrated to a simulation platform other than NeSTiNg, the save sub-module can be adapted to comply with the syntax of the simulation platform.

6.5 Evaluation

To evaluate the proposed framework we designed and configured a TSN network as illustrated in Figure 6.4, which is a hybrid ring-mesh topology connecting 12 end stations through 5 TSN switches. The network speed on each link is set to 1 Gbit/s. We assume that there are no processing delays in the TSN switches. Furthermore, the delays on the links themselves are considered negligible. We generated 40 flows of the scheduled traffic class in TSN. The periods of the flows are chosen from the set [1000 μs, 2000 μs, 5000 μs], which complies with the set of recommended periods in industrial automotive systems, as presented in the real-world automotive benchmarks [15]. We assume implicit deadlines for the flows, i.e., the deadline of each flow is equal to its period. The length of each frame in all the flows is considered to be equal to the maximum size of the Ethernet frame, i.e., 1542 Bytes. The use case topology is input to the schedule synthesis module that implements a schedule synthesis algorithm presented in [6]. This module is executed for approximately 20 hours on an HP Elite Book 820 running Ubuntu OS 18.04.4 LTS with CPU Core i5, 4 * 2.20 GHz Cores and 16 GB RAM.

We can take advantage of the GUI's visualization feature to view these large configuration files, as demonstrated in Figure 6.5. The results of the schedule synthesis module provide offsets to start the transmission of the flows. Hence, there are 40 offset values to set for each ST flow. In case there are multiple flows from a source end station to multiple destination end stations, we need to calculate the transmission offsets of frames transmitted from the same source end station up to the hyper period of the flows. Therefore, there are going to be 118 frame entries to configure all ST flow instances. Figure 6.5a shows the first instance of each flow from ES1. There are four flows from ES1 to the destination end stations ES7, ES8, ES9 and ES10 with periods 2000 μs, 1000 μs, 2000 μs and 5000 μs, respectively. On the link from ES1 to SW0, $[ES1, SW0]$, the offsets of frames are calculated for the duration of the hyper period, i.e., 10000 μs. In other words, it is required to calculate 10 offsets for a flow that has a period of 1000 μs within a hyper period of 10000 μs. Moreover, the switch gate states

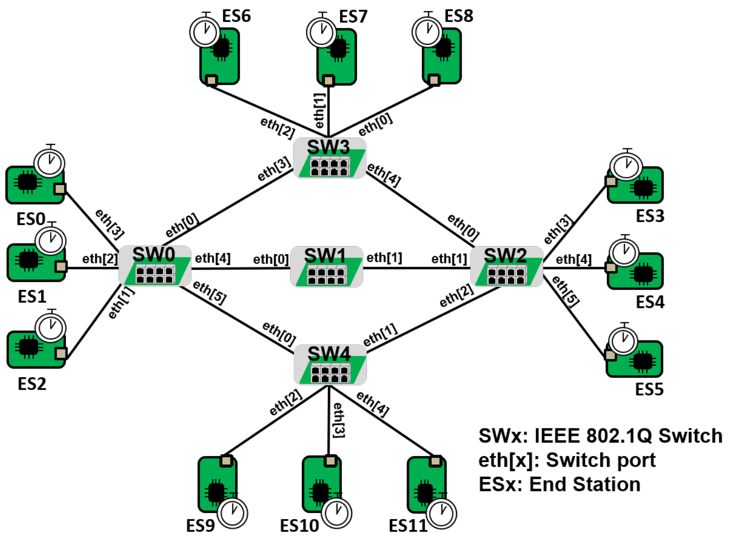


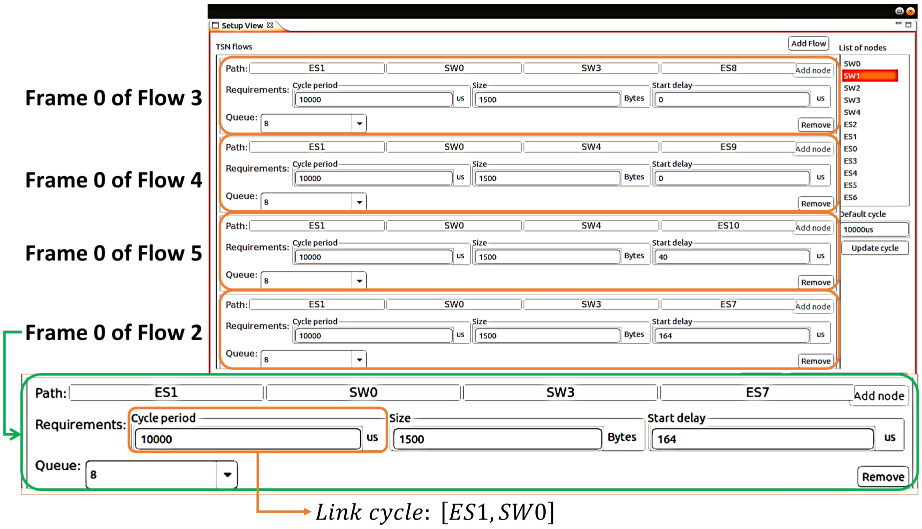
Figure 6.4. The use-case topology.

are specified by the scheduled time stamps to enable deterministic transmission of the ST flows from egress ports of the network switches. Table 6.1 shows the total number of gate entries to be configured in the switches for duration of the hyper period of each link. Since different ST flows with different profiles might be scheduled on the same link, the lengths, periods, and egress gate synthesis cycle should take hyper period of all the periodic ST flows on the link into account.

Table 6.1. Configuration complexity.

Switches	Total gate state changes
Switch 0	135
Switch 1	21
Switch 2	80
Switch 3	123
Switch 4	74

The next step, after obtaining the configuration parameters, is to insert them to the



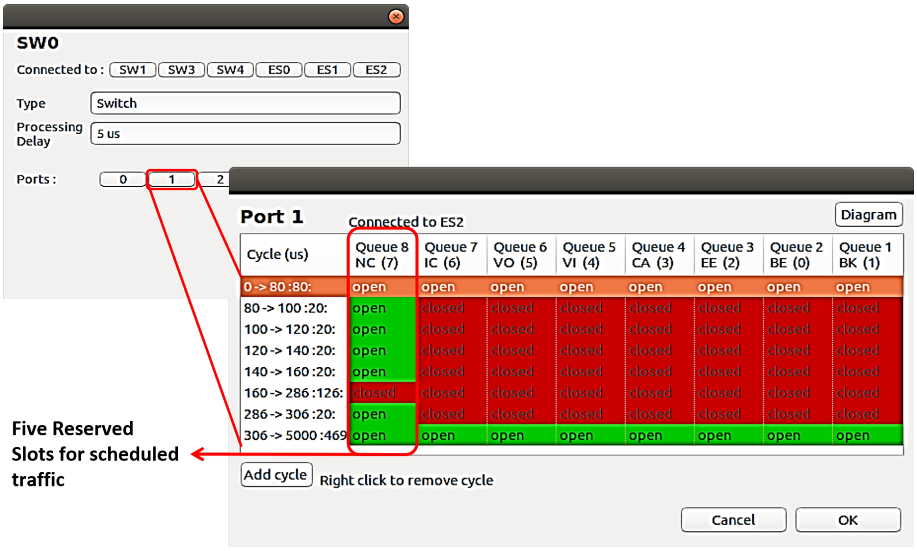
Path:	ES1	SW0	SW3	ES8
Requirements:	Cycle period 10000 us	Size 1500 Bytes	Start delay 0 us	
Queue:	8			

Path:	ES1	SW0	SW4	ES9
Requirements:	Cycle period 10000 us	Size 1500 Bytes	Start delay 0 us	
Queue:	8			

Path:	ES1	SW0	SW4	ES10
Requirements:	Cycle period 10000 us	Size 1500 Bytes	Start delay 40 us	
Queue:	8			

Path:	ES1	SW0	SW3	ES7
Requirements:	Cycle period 10000 us	Size 1500 Bytes	Start delay 164 us	
Queue:	8			

(a) Flow and path visualization with TSN Plug-in.


Five Reserved Slots for scheduled traffic
Diagram
Add cycle
Right click to remove cycle
Cancel
OK

SW0

Connected to : SW1 SW3 SW4 ES0 ES1 ES2

Type: Switch

Processing Delay: 5 us

Ports : 0 1 2

Port 1 Connected to ES2

Cycle (us)	Queue 8 NC (7)	Queue 7 IC (6)	Queue 6 VO (5)	Queue 5 VI (4)	Queue 4 CA (3)	Queue 3 EE (2)	Queue 2 BE (0)	Queue 1 BK (1)
0 -> 80 :80:	open	open	open	open	open	open	open	open
80 -> 100 :20:	open	closed	closed	closed	closed	closed	closed	closed
100 -> 120 :20:	open	closed	closed	closed	closed	closed	closed	closed
120 -> 140 :20:	open	closed	closed	closed	closed	closed	closed	closed
140 -> 160 :20:	open	closed	closed	closed	closed	closed	closed	closed
160 -> 286 :126:	closed	closed	closed	closed	closed	closed	closed	closed
286 -> 306 :20:	open	closed	closed	closed	closed	closed	closed	closed
306 -> 5000 :469	open	open	open	open	open	open	open	open

(b) Switch gate visualization by the GUI.

Figure 6.5. Configuration visualizations with TSN Plug-in.

topology created in the NeSTiNg simulator using the plug-in module. Accordingly, setting up the frame's offsets and gate states by the plug-in GUI is quite a tedious task, hence we use the plug-in's automated mode sub-module and apply the configuration data files from the traffic generator and schedule synthesis modules. Consequently, 867 lines for configuration of scheduled frames, 1964 lines of gate states configurations, and 77 lines for the switches' routing tables were automatically generated to setup the NeSTiNg simulator. Figure 6.5b demonstrates the gate states of port ”1” at the switch ”SW0” connected to ES2 up to the hyper period (5000 μs) of the link from SW0 to ES2. The Queue8 column in Figure 6.5b shows the reserved ST transmission slots on corresponding link. The size of the frame transmission slot is set to 20 μs because the complete transmission of an Ethernet frame on a link with 1 Gbit/s speed is 13 μs. Hence, we assume a 7 μs safety margin for the ST frame to pass through the link and the switch. We assume that open state is the default value of the switch gates. Furthermore, there are five 20 μs slots on the link, $[SW0, ES2]$, which are reserved for transmission of the five ST frames. This is the link where the traffic from the source end stations ES3, ES5, ES7, ES10, ES11 are transmitted, each with a period of 50000 μs.

The framework's final module, enables extraction of end-to-end delays of all ST flows after running the simulation for a desired amount of time. The feasibility of the schedules is inspected by checking if the maximum end-to-end delay of each flow is less than or equal to the flow period. Table 6.2 presents the maximum end-to-end delay for each ST flow retrieved by the results extraction sub-module, where TX and RX indicate the source end station and the destination end stations. The results confirm the feasibility and determinism of the offline schedule.

6.6 Conclusions

This paper proposed an automated modular framework to facilitate automated offline scheduling, pre-simulation configurations and interpretation of simulation results in TSN networks. The proposed framework addresses the challenge of configuring synthesized schedules to the state-of-the-art simulation framework, namely NeSTiNg. The proposed framework automatically translates the TSN flow schedules into the simulation platform's compatible syntax without any manual intervention. The proposed framework is implemented as an open-source TSN plugin, which is integrated to the NeSTiNg simulation framework. The applicability of the framework is demonstrated by automatically configuring a large TSN network and performing the simulation-based evaluation of the network. The evaluation results show the feasibility of the proposed framework for TSN networks. Furthermore, the results demonstrate the interoperability of the proposed framework with the state-of-the art simulation frameworks for TSN. The

Table 6.2. End-to-end delay per transmitted flow.

ID	TX	RX	Period(μs)	Delay(μs)
1	ES0	ES6	5000	52
2	ES1	ES7	2000	52
3	ES1	ES8	1000	52
4	ES1	ES9	2000	92
5	ES1	ES10	5000	37
6	ES2	ES3	1000	93
7	ES2	ES7	1000	82
8	ES3	ES1	5000	81
9	ES3	ES2	5000	64
10	ES3	ES4	2000	24
11	ES3	ES5	2000	37
12	ES4	ES3	5000	53
13	ES4	ES8	1000	65
14	ES4	ES9	1000	92
15	ES4	ES11	5000	53
16	ES5	ES0	2000	1888
17	ES5	ES2	5000	72
18	ES5	ES4	2000	37
19	ES5	ES6	5000	54
20	ES6	ES5	5000	64
21	ES6	ES7	5000	34
22	ES6	ES8	2000	24
23	ES7	ES1	5000	57
24	ES7	ES2	5000	77
25	ES7	ES4	1000	72
26	ES8	ES0	2000	158
27	ES8	ES10	1000	72
28	ES8	ES11	5000	75
29	ES9	ES0	5000	970
30	ES9	ES1	5000	53
31	ES9	ES3	5000	37
32	ES10	ES2	5000	61
33	ES10	ES5	5000	73
34	ES10	ES6	1000	112
35	ES10	ES7	5000	136
36	ES10	ES11	5000	37
37	ES11	ES2	5000	61
38	ES11	ES7	1000	97
39	ES11	ES9	5000	45
40	ES11	ES10	5000	49

benefits of the proposed automated configuration framework are magnified in the case of large TSN networks, where manual configurations can be error prone, time-consuming and very challenging to perform, thus rendering the existing methods impractical for industrial applications. As the future work, we aim at integrating the proposed framework with existing industrial tools for modeling of automotive embedded systems that use TSN for on-board network communication.

Acknowledgement

The work in this paper is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the projects DESTINE and PROVIDENT, and by the Swedish Knowledge Foundation (KKS) through the projects DPAC, HERO & FIESTA. The authors thank all industrial partners, especially Arcticus Systems, Volvo CE, and HIAB Sweden.

Bibliography

[1] IEEE Time-Sensitive Networking (TSN) Task Group.

[2] L. Lo Bello, R. Mariani, S. Mubeen, and S. Saponara. Recent advances and trends in on-board embedded and networked automotive systems. *IEEE Transactions on Industrial Informatics*, 2019.

[3] L. Lo Bello and W. Steiner. A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems. *Proceedings of the IEEE*, June 2019.

[4] Jonathan Falk, David Hellmanns, Ben Carabelli, Naresh Nayak, Frank Dürr, Stephan Kehrer, and Kurt Rothermel. NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++. In *Proceedings of the 2019 International Conference on Networked Systems*. IEEE, March 2019.

[5] David Hellmanns and Jonathan Falk. Nesting - network simulator for time-sensitive networking, 2020.

[6] Silviu S Craciunas, R Serna Oliver, and TC AG. An overview of scheduling mechanisms for time-sensitive networks. *Proceedings of the Real-time summer school (ETR)*, 2017.

[7] Hashemi Farzaneh et al. *A Modeling Framework to Facilitate Schedule Synthesis of Time-Sensitive Networking*. PhD thesis, Technische Universität München, 2019.

[8] Aellison Cassimiro T dos Santos, Ben Schneider, and Vivek Nigam. TSNSCHED: Automated schedule generation for time sensitive networking. In *2019 Formal Methods in Computer Aided Design*. IEEE, 2019.

[9] V. Gavriluţ, L. Zhao, M. L. Raagaard, and P. Pop. AVB-aware routing and scheduling of time-triggered traffic for TSN. *IEEE Access*, 2018.

[10] M. Pahlevan, R. Obermaisser. Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks. In *23rd International Conference on Emerging Technologies and Factory Automation*, 2018.

[11] J. Jiang, Y. Li, S. H. Hong, A. Xu, and K. Wang. A time-sensitive networking (TSN) simulation model based on OMNET++. In *2018 IEEE International Conference on Mechatronics and Automation*, 2018.

[12] OMNeT++ technical articles - result analysis with python, 2020.

[13] A Quick Overview of the OMNeT++ Intgrated Development Environment, 2020, `https://doc.omnetpp.org/omnetpp/IDE-Overview.pdf`.

[14] A. Bergström, Automatic Generation of Network Configuration in Simulated Time Sensitive Networking (TSN) Applications, Master Thesis, School of Innovation, Design and Engineering, Mälardalen University, Sweden, 2020.

[15] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real World Automotive Benchmarks for Free. In *6th Int. Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems*, 2015.

Chapter 7

Paper B: Synthesising Schedules to Improve QoS of Best-effort Traffic in TSN Networks

Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Saad Mubeen.
In the 29th International Conference on Real-Time Networks and Systems (RTNS), 2021.

Abstract

The IEEE Time-Sensitive Networking (TSN) standards' amendment 802.1Qbv provides real-time guarantees for Scheduled Traffic (ST) streams by the Time Aware Shaper (TAS) mechanism. In this paper, we develop offline schedule optimization objective functions to configure the TAS for ST streams, which can be effective to achieve a high Quality of Service (QoS) of lower priority Best-Effort (BE) traffic. This becomes useful if real-time streams from legacy protocols are configured to be carried by the BE class or if the BE class is used for value-added (but non-critical) services. We present three alternative objective functions, namely Maximization, Sparse and Evenly Sparse, followed by a set of constraints on ST streams. Based on simulated stream traces in OMNeT++/INET TSN NeSTiNg simulator, we compare our proposed schemes with a most commonly applied objective function in terms of overall maximum end-to-end delay and deadline misses of BE streams. The results confirm that changing the schedule synthesis objective to our proposed schemes ensures timely delivery and lower end-to-end delays in BE streams.

7.1 Introduction

Recent functionality advancements and innovation in the applications of embedded systems, especially in the automotive and automation domains, require high-bandwidth and low-latency communications. To meet these communication requirements, the IEEE 802.1 TSN task group[1] took an initiative to develop a set of Time-Sensitive Networking (TSN) standards over the switched Ethernet. Notable features of TSN include: clock synchronization (IEEE Std 802.1AS-2020) [1], Time-Aware Shaper (TAS) for offline scheduled time-triggered traffic (IEEE 802.1Qbv), Credit-based Shaper (CBS) and bandwidth reservation (IEEE Std 802.1Qav), frame preemption (IEEE Std 802.1Qbu), among others, which are rolled into the IEEE 802.1Q-2018 standard [2].

The queuing and forwarding mechanism in the TSN standards distinguishes between critical and non-critical traffic classes. The critical traffic classes are defined as Classes A and B. Where Class A has a higher priority than Class B. Whereas, the non-critical traffic class is defined as the best-effort (BE) traffic class with the lowest priority. In addition, the scheduled traffic (ST) enhancement in the TSN standard defines an ST traffic class with strict temporal isolation. The temporal isolation is achieved by a gate mechanism following the TAS mechanism, where the traffic transmission is allowed only when the gate for a queue is open. The gates operation follows a pre-defined gate control list (GCL) that repeats periodically defining which gate on the queues should be open at each time slot.

Currently the use of TSN in industrial systems is gaining a significant momentum [3, 4]. However, redesigning and replacing the existing communication systems is a relatively costly process. For example, many industrial systems use different Ethernet communication protocols, e.g., EtherCAT[2], with very different interfaces and message format compared to that of TSN. Therefore, one of the main challenges is to replace the existing networks with TSN making as little as possible changes in the end stations. Within this context, a non-trivial task is to map the existing network traffic to the TSN traffic classes such that the previous properties, e.g., jitter and delay, still hold. One of the straightforward solutions is to map all time-critical traffic into the ST traffic class to ensure the low jitter and low-latency transmission for the traffic. However, there are traffic that do not have strict deadlines to meet, yet achieving an acceptable level of Quality of Service (QoS) for them is expected. For example, diagnostic signals, network status checks and software update signals are among less time-critical traffic, which usually have soft real-time requirements. Frequent violation of the timing requirements for such traffic can hamper the corresponding functionalities. These type of traffic are often mapped to the BE class that requires no change in the network interfaces of the

[1] https://1.ieee802.org/tsn/

[2] https://www.ethercat.org/

end stations.

Many works in the literature address the problem of ST traffic scheduling, which is known as an NP-hard problem. Most of the solutions formulate the problem as an optimization problem (essentially a bin-packing problem) that is solved by defining a set of constraints, e.g., [5, 6]. The main requirement for the solver in these solutions is to provide a feasible schedule taking jitter and delay of the ST traffic into account. There are also a few works that consider different optimization goals such as maximizing porosity on the network links [7] to allow fast schedule updates in the case of faults in the network. There are a few solutions based on meta-heuristic algorithms, e.g., the solution in [8] that uses the genetic algorithm. Nevertheless, the proposed ST scheduling solutions in the literature strictly consider the timing requirements for the ST traffic, with an exception in [9] that considers schedulability of classes A and B while defining a routing mechanism for the ST traffic. To the best of our knowledge, none of the existing ST scheduling solutions take into account QoS for the BE traffic that has soft real-time requirements, e.g., legacy diagnostic signals or network status checks. In this paper, we propose a solution to synthesize feasible schedules for the ST traffic, while providing a high level of QoS for the BE traffic. To the extent of our knowledge, existing analytical schedulability analysis methods do not provide support for schedulability analysis of low priority BE traffic in IEEE TSN standards. Therefore, we use a simulation method to evaluate our proposed solutions.

The main contributions in this paper are as follows:

- we mathematically model and present optimization constraints to consider the QoS for the BE traffic;

- we propose new optimization objective functions to obtain a feasible ST schedule while improving the QoS of the BE traffic;

- we show that the commonly used objective function (minimizing the ST offsets) in the existing works leads to poor QoS for any lower priority traffic class; and

- we evaluate the proposed solutions with a set of experiments using the OMNeT++ simulation platform. We use Z3 SMT/OMT solver to implement the proposed constraints and objective functions. Then, we compare the proposed solutions with the existing solutions to show their effectiveness in providing a high level of QoS for the BE traffic.

7.2 Background and Related Work

7.2.1 The gate mechanism in TSN

The IEEE 802.1Qbv standard (rolled into IEEE 802.1Q-2018) defines a gate mechanism following the TAS mechanism to allow the ST traffic being transmitted without any interference. According to this mechanism, the class types are recognized by a 3-bit Priority Code Point (PCP) value in the IEEE TSN 802.1Q compatible frame headers. The example in Figure 7.1 illustrates a simplified gate mechanism, in which Q_0, Q_1, Q_2 and Q_3 correspond to classes ST, A, B and BE, respectively. A Gate Control List (GCL) contains an array of time-stamped vectors to control the output of each queue. The list is repeating periodically. At the specified times in the time-stamped vector, the data in the vector is sent to the gate drivers, which enable or disable the transmission of the associated class. In this example, the value `$1$` represents open state to enable transmission, while `$0$` closes the gate to disable the transmission in certain queues. Note that only classes A and B undergo the CBS shaper.

7.2.2 Schedule Synthesis

The schedule synthesis approaches can be grouped into two branches as shown in Figure 7.2: (i) Satisfiability Modulo Theorems (SMT) and Optimization Modulo Theorem (OMT), and (ii) heuristics and meta-heuristic approaches. Most of the works in schedule synthesis were using SMT solvers to search all the possibilities for optimal schedule. SMT solvers are automated theorem provers to prove satisfiability and validity of first-order logical statements by examining each possible combination of variables in the search space. The output of the SMT solvers is a random value for the SMT variable that satisfies the specified constraints. To improve scalability, some works have either used a new type of optimized SMT solver called OMT or Mixed-Integer Programming (MIP). The OMT Solvers have become popular as they enable optimizing SMT Solver variables based on user-defined objective functions and allow to solve linear optimization problems over satisfiable SMT formulas [10].

The majority of the previous works were concentrated on the synthesis of feasible schedules for the high priority critical traffic based on a work by Steiner et al. [11]. In the series of works by Craciunas et al. [12, 5, 13, 14, 15, 16] the network was modelled by directed graphs that indicate relations between end stations and switches via links. In [5], schedule optimization constraints were derived from TTEthernet scheduling constraints to address the specifications of IEEE 802.1Qbv. The generic constraints of the solution include: frame, link, stream and end-to-end delay timing constraints. Schneider et al. [17, 18, 19] introduced an SMT-based ST schedule synthesizer tool,

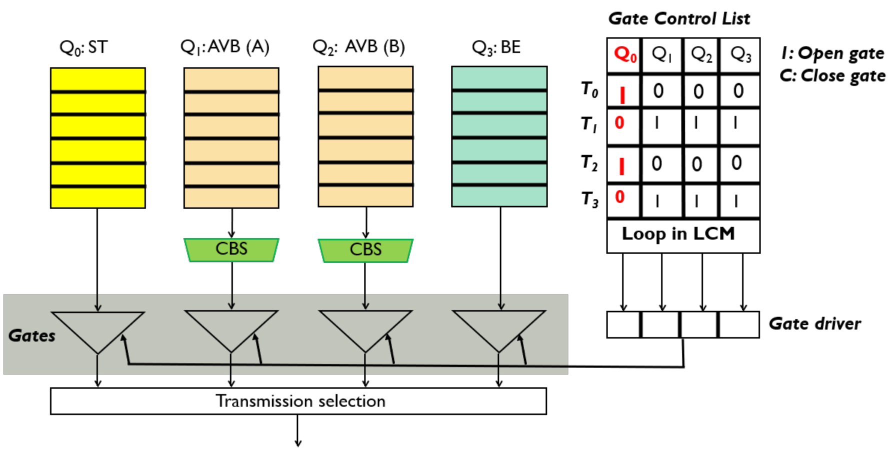


Figure 7.1. Gate mechanism block diagram.

named TSNSched, which modeled TSN network via data structures.

Steiner et al.[20] introduced five strategies to enable porous (or blank) schedule synthesis. These strategies include: a) a priori schedule variation to force blank schedules, b) a posteriori schedule variation with less computational time-complexity, c) combined schedule variation in order to assure the least overhead, d) interpretation approach to change the timeline into equal time slots defining minimum size for blanks, and e) combined schedule variation with the above steps [20].

In the same line of research, the work by Pozo et al. [7] investigated the generation of porous link schedules, which can help in achieving extra time spans to react to possible link failures. The proposed method takes advantage of a link reparation post-processing procedure to find alternative paths that do not pass through the failed links. In addition, the work by Gavrilut et al. [9] proposed a two-stage approach for scheduling ST traffic considering an optimum routing for the ST traffic, while taking into account the schedulability of the AVB traffic (i.e., A and B).

A series of works considered different approaches than only fitting the ST traffic into time-slots. For example, the work by Reusch et al. [6] proposed a window-based schedule synthesis approach that maximizes the bandwidth allocated for lower priorities. Their solutions require exclusion of deadline miss constraints for the ST schedule. Therefore, a post-processing iterative optimization heuristic was presented to shift slack windows in a manner to save deadline misses due to schedules made by a gap synthesis. Thereafter, the proposed method is evaluated by the worst-case

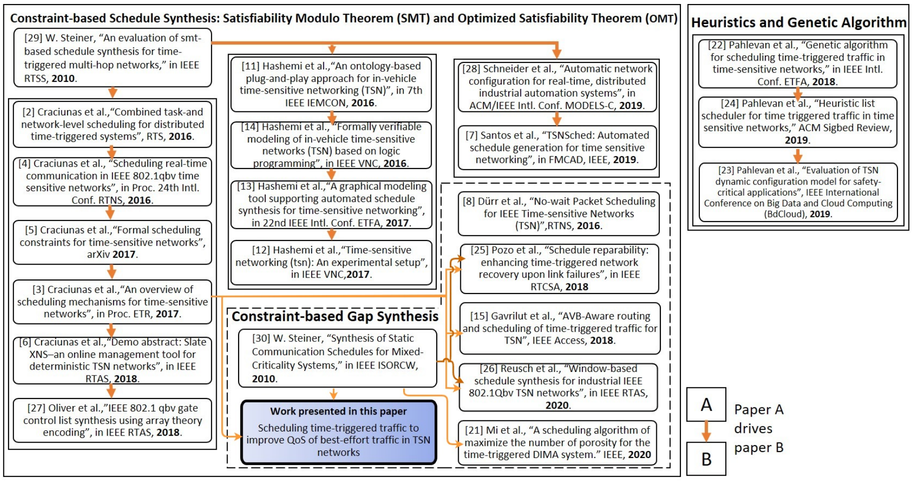

Constraint-based Schedule Synthesis: Satisfiability Modulo Theorem (SMT) and Optimized Satisfiability Theorem (OMT)
[29] W. Steiner, "An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks," in IEEE RTSS, 2010.
[2] Craciunas et al.,"Combined task-and network-level scheduling for distributed time-triggered systems", RTS, 2016.
[4] Craciunas et al., "Scheduling real-time communication in IEEE 802.1qbv time sensitive networks", in Proc. 24th Intl. Conf. RTNS, 2016.
[5] Craciunas et al.,"Formal scheduling constraints for time-sensitive networks", arXiv 2017.
[3] Craciunas et al.,"An overview of scheduling mechanisms for time-sensitive networks", in Proc. ETR, 2017.
[6] Craciunas et al.,"Demo abstract: Slate XNS–an online management tool for deterministic TSN networks", in IEEE RTAS, 2018.
[27] Oliver et al.,"IEEE 802.1 qbv gate control list synthesis using array theory encoding", in IEEE RTAS, 2018.
[11] Hashemi et al.,"An ontology-based plug-and-play approach for in-vehicle time-sensitive networking (TSN)", in 7th IEEE IEMCON, 2016.
[14] Hashemi et al., "Formally verifiable modeling of in-vehicle time-sensitive networks (TSN) based on logic programming", in IEEE VNC, 2016.
[13] Hashemi et al.,"A graphical modeling tool supporting automated schedule synthesis for time-sensitive networking", in 22nd IEEE Intl. Conf. ETFA, 2017.
[12] Hashemi et al.,"Time-sensitive networking (tsn): An experimental setup", in IEEE VNC,2017.
[28] Schneider et al., "Automatic network configuration for real-time, distributed industrial automation systems", in ACM/IEEE Intl. Conf. MODELS-C, 2019.
[7] Santos et al., "TSNSched: Automated schedule generation for time sensitive networking", in FMCAD, IEEE, 2019.
[8] Dürr et al., "No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN)",RTNS, 2016.
[25] Pozo et al., "Schedule reparability: enhancing time-triggered network recovery upon link failures", in IEEE RTCSA, 2018
[15] Gavrilut et al., "AVB-Aware routing and scheduling of time-triggered traffic for TSN", IEEE Access, 2018.
[26] Reusch et al., "Window-based schedule synthesis for industrial IEEE 802.1Qbv TSN networks", in IEEE RTAS, 2020.
[21] Mi et al., "A scheduling algorithm of maximize the number of porosity for the time-triggered DIMA system." IEEE, 2020
Constraint-based Gap Synthesis
[30] W. Steiner, "Synthesis of Static Communication Schedules for Mixed-Criticality Systems," in IEEE ISORCW, 2010.
Work presented in this paper
Scheduling time-triggered traffic to improve QoS of best-effort traffic in TSN networks
Heuristics and Genetic Algorithm
[22] Pahlevan et al., "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in IEEE Intl. Conf. ETFA, 2018.
[24] Pahlevan et al., "Heuristic list scheduler for time triggered traffic in time sensitive networks," ACM Sigbed Review, 2019.
[23] Pahlevan et al., "Evaluation of TSN dynamic configuration model for safety-critical applications", IEEE International Conference on Big Data and Cloud Computing (BdCloud), 2019.
A
B
Paper A drives paper B

Figure 7.2. Overview of the schedule synthesis approaches.

delay analysis proposed in [21]. Another similar work by Mi et al. [22] considered two-stage gap synthesis. The purpose of the work is to create porous schedules to reduce the transmission delay and jitter of other real-time traffic. Moreover, the works by Hashemi et al. [23, 24, 25, 26, 27] adopted an ontology-based approach, utilizing logic programming in Prolog programming language, to show properties and the relations between network entities and components.

Considering the solutions based on heuristics and meta-heuristic we can find several works in the research community. For instance, the work by Pahlevan et al. [8, 28, 29] investigated heuristic/GA-based solutions to overcome time consuming constraint solving processes. However, the proposed solution was not compared with the SMT-based solutions. Moreover, Dürr et al. [30] proposed a Tabu search-based approach to find feasible schedules of the ST traffic. The authors proposed a bandwidth utilization by means of scheduling the ST frames close to each other.

Although there are several solutions to schedule ST traffic in TSN networks, as reviewed above, most of them considered only the timing properties of the ST traffic in their scheduling solutions. The exception is the work in [9] and [7], where in the former the schedulability of AVB traffic was considered and in the latter repairability of the links after failures was the main aim. In our work, we propose a set of constraints and three objective functions to schedule the ST traffic while improving the QoS of the BE traffic in the network considering that they have soft real-time properties due to legacy network support. We base our solution on the work presented in [5].

7.3 System Model

The system model considered in this paper is inspired by the system model in [5]. Note that the paper focuses only on the ST and BE traffic classes. The network model, as shown in Figure 7.3, is represented by a directed graph, $G(V, L)$. The vertices V represent end stations and switches in the network. The links, represented by L, are modeled as two-directional edges connecting a set of vertices. A two-directional link between the end stations $v_a \in V$ and $v_b \in V$ is represented by $[v_a, v_b] \in L$. The link attributes are denoted by the tuple $\langle speed, d, mt\rangle$, where the parameters $speed$, d and mt indicate the link's speed, link propagation delay and macrotick, respectively. The model is flexible such that each link can accept different speed and propagation delay. The macrotick serves for the scalar granularity of the schedule time-line. For example, macrotick of $1\mu s$ indicates that the schedule time unit is in the order of microseconds.

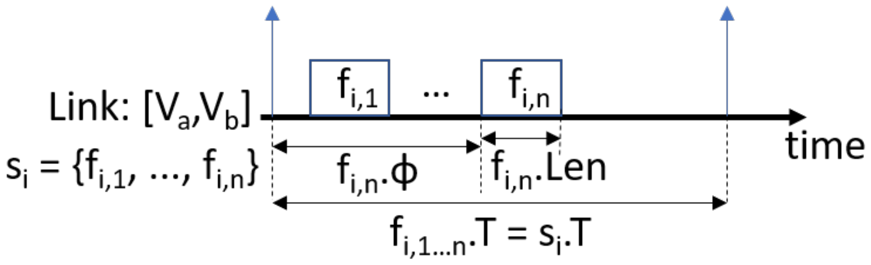


Figure 7.3. System model parameters.

The set of streams in the network are represented by S. A uni-cast stream with ID i belonging to S is represented by s_i, i.e., $s_i \in S$. The path taken by a stream from the source end station, v_s, to the destination end station, v_d, through a set of switches is shown by $s_i.path = [[v_s, v_{s+1}], \ldots, [v_{d-1}, v_d]]$, where $[v_s, v_{s+1}]$ and $[v_{d-1}, v_d]$ indicate the first and last links along the path of the stream. The stream's attributes are defined by the tuple, $\langle e2e, Size, T, p\rangle$. Where, $e2e$ is the timing constraint on the end-to-end delay of the stream, i.e., the maximum end-to-end delay shall not exceed the value assigned to the e2e attribute. $Size$ is transmission time of the stream on the link, which is the time it takes to transmit a full stream on the link in the scale of macroticks. The parameters, T and p, are stream's period and priority, respectively.

The parameter, p, obtains a value from the set $p = \{BE, ST\}$, where BE shows the best-effort class and ST shows the ST class. Note that in case of BE class, T is the minimum inter-arrival time and $e2e$ is the deadline for the BE stream.

A stream may contain n number of frames, i.e., $s_i = \{f_{i,1}, \ldots, f_{i,n}\}$. A frame at the hop between the end station or switch v_a and the end station or switch v_b is represented by $f_{i,j}^{[v_a,v_b]}$, where the first subscript i represents the stream ID and the second subscript j represents the frame ID. Because a stream may contain large data, according to the Ethernet protocols, the data has to be fragmented into frames. The maximum allowed Ethernet frame is limited by the Maximum Transmission Unit (MTU). In this model, each stream can be fragmented into several frames. Note that if the stream size does not exceed the MTU size, then all data is fitted in one frame. Each frame has the attributes $\langle T, Len, \phi \rangle$, where T is the frame's period inherited from the parent stream. The transmission time of the frame over the link $[v_a, v_b]$ is represented by $f_{i,j}^{[v_a,v_b]}.Len$. If a stream belongs to the class ST, $f_{i,j}^{[v_a,v_b]}.\phi$ represents the offset of j^{th} frame of the i^{th} stream on the link $[v_a, v_b]$. There are no offsets for BE frames, hence ϕ is not applicable for the BE frames. Moreover, each BE frame is assumed to arrive at the start of its period, which in turn generates the worst-case situation for the BE frames. The parameters for the frames belonging to a ST stream and their attributes are shown in Figure 7.3 for one period of the stream s_i over one link $[v_a, v_b]$.

7.4 Network Constraints

In this section, we first give an overview of the existing optimization to schedule ST traffic in TSN networks, then we present the proposed additional constraints to improve the QoS of the BE traffic in TSN networks.

7.4.1 Existing Constraints

We base our work on the ST schedule synthesis constraints in [5] that defines six different constraints to find a feasible schedule. These constraints include: (i) frame constraint, (ii) link constraint, (iii) stream constraint, (iv) end-to-end constraint, (v) stream isolation constraint, and (vi) frame isolation constraint. Below, we present these constraints in detail.

Frame Constraint

This constraint ensures that the offset of each ST frame is at or after time zero. Also, the transmission of the frame must be finished at or before the start of its next period.

$$\begin{aligned}&\forall s_i \in S, s_i.p = ST, \forall[V_a, V_b] \in L, \forall f_{i,j}^{[V_a,V_b]} \in s_i^{[V_a,V_b]} : \\ &(f_{i,j}^{[V_a,V_b]}.\phi >= 0) \wedge (f_{i,j}^{[V_a,V_b]}.\phi <= f_{i,j}^{[V_a,V_b]}.T - f_{i,j}^{[V_a,V_b]}.Len)\end{aligned} \tag{7.1}$$

Link Constraint

The constraint on links guarantees that only one frame at a time can be transmitted on a link within the path taken by the stream;

$$\begin{aligned}&\forall s_i, s_j \in S, s_i.p = ST, s_j.p = ST, i \neq j, \forall[V_a, V_b] \in L, \\ &\forall f_{i,k}^{[V_a,V_b]} \in s_i^{[V_a,V_b]}, \forall f_{j,l}^{[V_a,V_b]} \in s_j^{[V_a,V_b]}, \\ &\forall \alpha \in [0, \tfrac{lcm(s_i.T, s_j.T)}{s_i.T} - 1], \forall \beta \in [0, \tfrac{lcm(s_i.T, s_j.T)}{s_j.T} - 1] : \\ &(f_{j,l}^{[V_a,V_b]}.\phi + (\beta \times f_{j,l}^{[V_a,V_b]}.T) + f_{j,l}^{[V_a,V_b]}.Len \leq \\ &f_{i,k}^{[V_a,V_b]}.\phi + (\alpha \times f_{i,k}^{[V_a,V_b]}.T)) \vee \\ &(f_{i,k}^{[V_a,V_b]}.\phi + (\alpha \times f_{i,k}^{[V_a,V_b]}.T) + f_{i,k}^{[V_a,V_b]}.Len \leq \\ &f_{j,l}^{[V_a,V_b]}.\phi + (\beta \times f_{j,l}^{[V_a,V_b]}.T))\end{aligned} \tag{7.2}$$

where, α and β are indices to calculate the offsets of multiple frames within the hyper periods of the streams. The hyper period is calculated by the function $lcm()$ that receives the streams pair by pair as inputs and gives the least common multiple of the frames' periods.

Stream Constraint

In order to constrain the orderly transmission and reception of frames within a stream through the path, the sequence of the frames should be considered according to Eq. (7.3);

$$\begin{aligned}&\forall s_i \in S, s_i.p = ST, \forall[V_a, V_x], [V_x, V_b] \in L, \\ &f_{i,j}^{[V_a,V_x]} \in s_i^{[V_a,V_x]}, f_{i,j}^{[V_x,V_b]} \in s_i^{[V_x,V_b]} : \\ &((f_{i,j}^{[V_x,V_b]}.\phi \times [V_x, V_b].mt) - [V_x, V_b].d - \delta) \geq \\ &((f_{i,j}^{[V_a,V_x]}.\phi + f_{i,j}^{[V_a,V_x]}.Len) \times [V_a, V_x].mt)\end{aligned} \tag{7.3}$$

The stream constraint is dependent on synchronization of clocks in the source and destination end stations. Where, δ is the synchronization factor, denoting the worst-case drift between the clocks. We assume identical clocks in the source and destination end stations.

End-to-end Constraint

In order to meet the timing requirements of ST streams, the ST streams must not be delivered after their deadlines. Therefore, we need to ensure that the stream's last frame is delivered to the destination end station before the end-to-end deadline. Additionally, the constraint in Eq. (7.4) takes into account the delay of the links in the path to the destination end station and the worst-case difference between macro ticks of the local clocks of the source and destination end stations. Note that $f_{i,1}$ and $f_{i,|s_i|}$ denote the first and last frames in the stream, s_i. Moreover, $[v_s, v_{s+1}]$ defines the first link in the stream from the source while $[v_{d-1}, v_d]$ represents the last link in the stream to the destination end stations. Hence, $f_{i,|s_i|}^{[v_{d-1},v_d]}$ represents the last frame in the stream s_i at the destination end station.

$$\begin{aligned}
&\forall s_i \in S, s_i.p = ST, f_{i,j} \in s_i : \\
&(f_{i,1}^{[v_s,v_{s+1}]}.\phi \times [v_s, v_{s+1}].mt) + s_i.e2e \geq \\
&(f_{i,|s_i|}^{[v_{d-1},v_d]}.\phi + f_{i,|s_i|}^{[v_{d-1},v_d]}.Len) \times [v_{d-1}, v_d].mt
\end{aligned} \tag{7.4}$$

Stream Isolation Constraint

The transmission of multiple streams via the same queue can cause variations in the arrival times of the frames belonging to different streams. To ensure deterministic arrival of the streams, the offsets of all frames in a stream must be constrained to be less than the offset of the first frame in every other stream. Figure 7.4 shows an example of the case where stream isolation constraint is applicable. In this example, two streams s_i and s_j are transmitted to the end station v_a forwarding to the same end station v_b. The stream isolation constraint ensures that when the first frame of a stream, s_j in this example, is scheduled for transmission no other frames of other streams, s_i in this example, is scheduled until all frames of s_j are dispatched. In Figure 7.4, for simplicity of illustration we assumed that the periods of s_i and s_j are the same, hence the hyper period of them are the same as their periods. This constraint is presented in Eq. (7.5).

$$
\begin{aligned}
&\forall[V_a,V_b] \in L, \forall s_i^{[V_a,V_b]} \in S, s_j^{[V_a,V_b]} \in S, s_i.p = ST, s_j.p = ST, i \neq j,\\
&f_{i,k}^{[V_a,V_b]} \in s_i^{[V_a,V_b]}, f_{j,l}^{[V_a,V_b]} \in s_j^{[V_a,V_b]},\\
&\forall \alpha \in [0, \tfrac{lcm(s_i.T, s_j.T)}{s_i.T} - 1], \forall \beta \in [0, \tfrac{lcm(s_i.T, s_j.T)}{s_j.T} - 1]:\\
&((f_{j,|s_j|}^{[V_a,V_b]}.\phi \times [V_a,V_b].mt) + f_{j,|s_j|}^{[V_a,V_b]}.Len + (\beta \times s_j.T) + \delta \leq\\
&(f_{i,1}^{[V_x,V_a]}.\phi \times [V_x,V_a].mt) + (\alpha \times s_i.T) + [V_x,V_a].d) \vee\\
&((f_{i,|s_i|}^{[V_a,V_b]}.\phi \times [V_a,V_b].mt) + f_{i,|s_i|}^{[V_a,V_b]}.Len + (\alpha \times s_i.T) + \delta \leq\\
&(f_{j,1}^{[V_y,V_a]}.\phi \times [V_y,V_a].mt) + (\beta \times s_j.T) + [V_y,V_a].d)
\end{aligned}
\tag{7.5}
$$

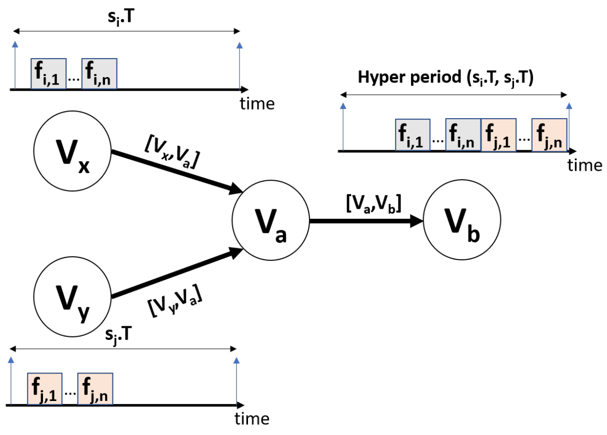


Figure 7.4. Stream isolation constraint.

Frame Isolation Constraint

In order to relax the stream isolation constraint by allowing frame interleaving between different streams an alternative constraint can be defined. The frame isolation constraint allows interleaving of frames from different streams considering that there are only frames of one stream in the queue at a time. When two source end stations V_x and V_y transmit streams via one or more shared switches within their path, the interleaving of frames in the same queue of the shared switch can cause non-deterministic delays of frames. For example, a stream with a higher period can cause long delays to a shorter period stream in case interleaving of frames occurs between these two streams. The frame isolation constraint, shown in Eq. (7.6), isolates the transmission of every frame of different streams in the queue. In other words, it guarantees that the offset plus the transmission time of each frame belonging to a stream (s_i transmitted from node V_x) is smaller than or equal to the offset of each frame belonging to another stream (s_j transmitted from node V_y) or vice versa. This scenario is depicted in Figure 7.5 assuming that both s_i and s_j have the same periods, hence the same hyper period on link $[v_a, v_b]$ with two possible schedules where one of them shows an interleaving of frames from the two streams.

$$\begin{aligned}
&\forall [V_a, V_b] \in L, \forall s_i^{[V_a,V_b]} \in S, s_j^{[V_a,V_b]} \in S, s_i.p = ST, s_j.p = ST, i \neq j,\\
&f_{i,k}^{[V_a,V_b]} \in s_i^{[V_a,V_b]}, f_{j,l}^{[V_a,V_b]} \in s_j^{[V_a,V_b]},\\
&\forall \alpha \in [0, \tfrac{lcm(s_i.T, s_j.T)}{s_i.T} - 1], \forall \beta \in [0, \tfrac{lcm(s_i.T, s_j.T)}{s_j.T} - 1] :\\
&((f_{j,l}^{[V_a,V_b]}.\phi \times [V_a, V_b].mt) + f_{j,l}^{[V_a,V_b]}.Len + (\beta \times s_j.T) + \delta \leq\\
&(f_{i,k}^{[V_x,V_a]}.\phi \times [V_x, V_a].mt) + (\alpha \times s_i.T) + [V_x, V_a].d) \vee\\
&((f_{i,k}^{[V_a,V_b]}.\phi \times [V_a, V_b].mt) + f_{i,k}^{[V_a,V_b]}.Len + (\alpha \times s_i.T) + \delta \leq\\
&(f_{j,l}^{[V_y,V_a]}.\phi \times [V_y, V_a].mt) + (\beta \times s_j.T) + [V_y, V_a].d)
\end{aligned} \tag{7.6}$$

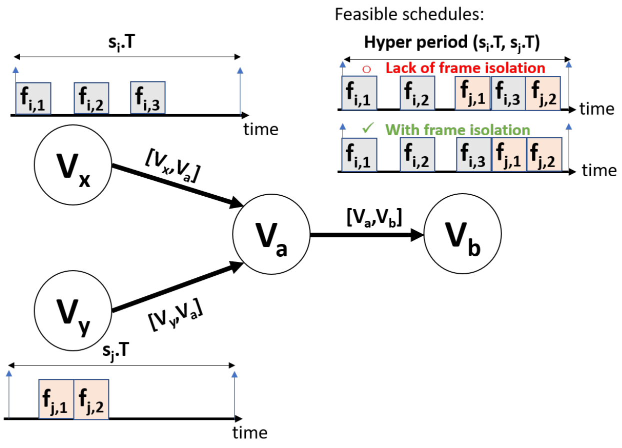


Figure 7.5. Frame isolation constraint.

7.4.2 Proposed Constraints

In this section, the problem that motivates the contributions of this paper is presented by an example, shown in Figure 7.6. In Figure 7.6(a) time stamps $[0, T_0]$, $[T_1, T_2]$ and $[T_3, T_4]$ are reserved for ST frames. The time slots represented by white boxes in Figure 7.6(a) indicate the times when the gates are open for transmission of BE frames. Based on this schedule, ST frames that are packed subsequently within the time slot $[T_1, T_2]$, cause deadline miss for the BE frame, $BE3$. On the other hand, Figure 7.6(b) shows the sparsification of ST frames within time stamp $[T_1, T_2]$ from the previous example into new scheduled time stamps $[T_1, T_2]$, $[T_3, T_4]$ and $[T_5, T_6]$. Consequently, with a few more gate state changes, the BE frame meets its deadline.

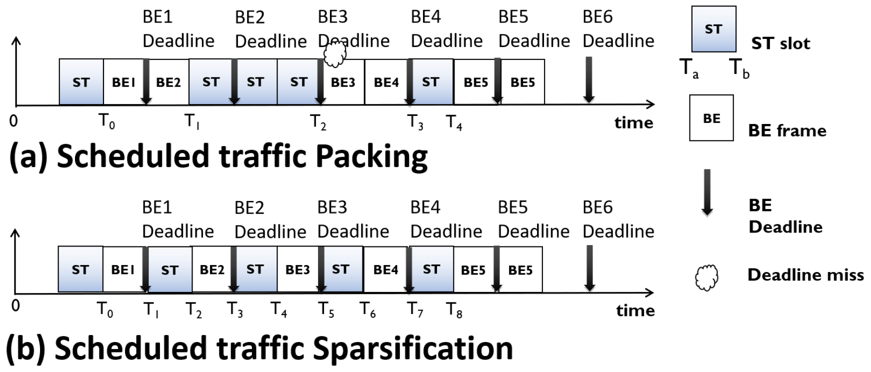


Figure 7.6. Example of schedule bin-packing and the effect on schedulability of BE traffic.

In order to obtain sparsity in the ST schedules in favor of BE frames, we propose a notion of *slack* after each ST frame transmission. Slack is a time interval after transmission of a ST frame, during which no ST frame can occupy the link's bandwidth. This prevents back-to-back transmission of ST frames [20]. We denote the slack per frame per link by $f_{i,j}^{[v_a,v_b]}.slack$. Figure 7.7 illustrates the proposed slack property of a frame per link. The goal is to find a feasible ST schedule such that a desired slack per frame per link is obtained. The realization of slacks for all frames leads to implementing spaces between ST frames. The size of slack is defined by the proposed constraints.

Porous Link Constraint

In order to incorporate the slack, the existing link constraints should be modified. The link constraint presented in Eq. (7.2) only restricted the timing overlap and placement of

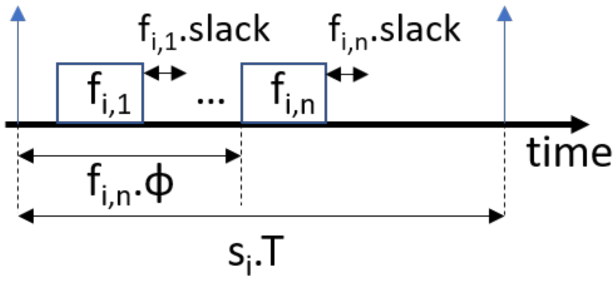


Figure 7.7. Illustration of the proposed slack property of a frame per link.

subsequent frames on the link. The constraint presented in Eq. (7.2) should be adapted into Eq. (7.7); the reason for this modification is that the original link constraint only restricted the timing overlap and subsequent placement of frames on the link.

$$\begin{aligned}
&\forall s_i, s_j \in S, s_i.p = ST, s_j.p = ST, \forall [V_a, V_b] \in L,\\
&\forall s_i^{[V_a,V_b]}, s_j^{[V_a,V_b]}, i \neq j, f_{i,k}^{[V_a,V_b]} \in s_i^{[V_a,V_b]}, f_{j,l}^{[V_a,V_b]} \in s_j^{[V_a,V_b]},\\
&\forall \alpha \in [0, \tfrac{lcm(s_i.T, s_j.T)}{s_i.T} - 1], \forall \beta \in [0, \tfrac{lcm(s_i.T, s_j.T)}{s_j.T} - 1]:\\
&(f_{j,l}^{[V_a,V_b]}.\phi + (\beta \times f_{j,l}^{[V_a,V_b]}.T) + f_{j,l}^{[V_a,V_b]}.Len + f_{j,l}^{[V_a,V_b]}.slack \leq\\
&f_{i,k}^{[V_a,V_b]}.\phi + (\alpha \times f_{i,k}^{[V_a,V_b]}.T))\vee\\
&(f_{i,k}^{[V_a,V_b]}.\phi + (\alpha \times f_{i,k}^{[V_a,V_b]}.T) + f_{i,k}^{[V_a,V_b]}.Len + f_{i,k}^{[V_a,V_b]}.slack \leq\\
&f_{j,l}^{[V_a,V_b]}.\phi + (\beta \times f_{j,l}^{[V_a,V_b]}.T))
\end{aligned} \tag{7.7}$$

Slack Size Constraint

The slack size constraint asserts the allowed slacks size for each frame scheduled on the link. The slack must be greater than or equal to zero but less than or equal to the difference between the frame period and its transmission time as follows:

$$\begin{aligned}
&\forall s_i \in S, s_i.p = ST, \forall [V_a, V_b] \in L, \forall f_{i,j}^{[V_a,V_b]} \in s_i^{[V_a,V_b]}:\\
&(f_{i,j}^{[v_a,v_b]}.slack \geq 0)\wedge\\
&(f_{i,j}^{[v_a,v_b]}.slack \leq f_{i,j}^{[v_a,v_b]}.T - f_{i,j}^{[v_a,v_b]}.Len)
\end{aligned} \tag{7.8}$$

The size of slacks on a link must be correlated with the number of ST frames scheduled on the link. Eq. (7.9) is the summation formulation of the frame slacks on each link. Figure 7.8 presents an example of the slack spaces after two frames from different streams scheduled on the link $[V_a, V_b]$.

$$\forall s_i \in S, s_i.p = ST, \forall [V_a, V_b] \in L, \forall f_{i,j}^{[V_a,V_b]} \in s_i^{[V_a,V_b]} : \\ hopSum^{[v_a,v_b]} = \sum f_{i,j}^{[v_a,v_b]}.slack \tag{7.9}$$

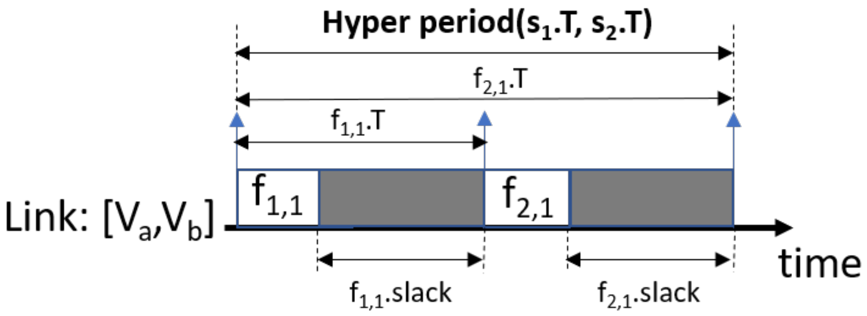


Figure 7.8. The allowed range for slacks on a link.

Hop Slacks Constraint

In order to bound the total amount of slacks allowed on the link, the constraints in Eq. (7.10) and Eq. (7.12) enforce the valid ranges of $hopSum$. The $minPorosity$ parameter is considered as the lower bound for $hopSum$ summation formulation, as shown in Eq. (7.10).

$$\forall s_i \in S, s_i.p = ST, \forall [V_a, V_b] \in L, \forall f_{i,j}^{[V_a,V_b]} \in s_i^{[V_a,V_b]} : \\ hopSum^{[v_a,v_b]} \geq minPorosity \tag{7.10}$$

where, $minPorosity$ can be any value from 0, however, if the value is selected between 0 and the frame size, assuming that the preemption is disabled, the slack after each frame is not enough for transmission of any BE frames.

Moreover, the sparse spaces on each link need to be specified per link, since the load of BE streams, converged with ST frames, varies on different links. The ST load on the link $[v_a, v_b]$ is specified by $[v_a, v_b].util_{ST}$ and is calculated by the equation:

$$\begin{aligned}&\forall [V_a, V_b] \in L, \forall s_i \in S, s_i.p = ST : \\ &[V_a, V_b].util_{ST} = \sum \frac{s_i^{[v_a,v_b]}.Size}{s_i^{[v_a,v_b]}.T}\end{aligned} \tag{7.11}$$

Furthermore, the upper bound of the $hopSum$ is presented in another constraint to ensure that slacks do not cause the porous link schedule to exceed the hyper period. Eq. (7.12) ensures that $hopSum$ is less than or equal to the total load that is left after the scheduled time for the ST frames on the link.

$$\begin{aligned}&\forall s_i \in S, s_i.p = ST, \forall [V_a, V_b] \in L, \forall f_{i,j}^{[V_a,V_b]} \in s_i^{[V_a,V_b]} : \\ &hopSum^{[v_a,v_b]} \leq lcm(S) \times (1 - [v_a, v_b].util_{ST})\end{aligned} \tag{7.12}$$

Equal Slack Constraint

Equal slack is an optional constraint to evenly distribute slacks on a link by adjusting equal slack sizes, as defined by Eq. (7.13). Note that the optimization time can be shortened by constraining equal-sized slacks due to limiting the options for the ST schedules. As a result, this constraint enables a selection trade-off between the freedom to choose ST schedules, enhancing arrival of periodic BE and optimization time.

$$\begin{aligned}&\forall s_i, s_j \in S, s_i.p = ST, s_j.p = ST, \forall [V_a, V_b] \in L, \\ &\forall f_{j,l}^{[V_a,V_b]} \in s_j^{[V_a,V_b]}, \forall f_{i,k}^{[V_a,V_b]} \in s_i^{[V_a,V_b]}, i \neq j : \\ &f_{i,k}^{[v_a,v_b]}.slack = f_{j,l}^{[v_a,v_b]}.slack\end{aligned} \tag{7.13}$$

7.4.3 Objective Functions

Minimizing the offsets of the ST frames' offsets is the objective function that is considered in all the previous works concerning the schedule synthesis, such as the work in [30]. The minimization objective function, given in Eq. (7.14), generates offsets that pack the ST frames to the beginning of the schedule subject to the generic constraints presented in Section 7.4.1.

$$\begin{aligned}&minimize \sum_{\forall [v_a,v_b] \in L} \sum_{\forall s_i \in S} \sum_{\forall f_{i,j} \in s_i} f_{i,j}^{[v_a,v_b]}.\phi \\ &\text{subject to} : \{(7.1), (7.2), (7.3), (7.4), (7.5) OR (7.6)\}\end{aligned} \tag{7.14}$$

The minimization approach can be convenient to secure both schedulability and timeliness of ST traffic. Some of the lower priority streams, including the BE, may be constrained by timing requirements. In order to address these requirements, three new objective functions are proposed as alternatives to the widely used minimization optimization objective function.

Maximization

This optimization objective function still packs ST frames together, but in contrast to the minimization objective function, it schedules the transmission of the ST frames as close as possible to their deadlines. As the number of ST frames increases, the ST reserved time slots on the link also increases. This can increase the maximum end-to-end delay of the BE frames, which in turn, may cause deadline misses. If we maximize the ST schedules, we allow the majority of open bandwidth for BE frames in the beginning of the schedule, increasing the likelihood of transmission of the BE frames before the scheduled transmission of the ST frames. This can improve the schedulability of the BE frames that may have deadlines shorter than those of the ST frames. Eq. (7.15) defines the maximization objective function subject to the generic constraints presented in Section 7.4.1.

$$\begin{aligned} &maximize \sum_{\forall [v_a,v_b]\in L} \sum_{\forall s_i \in S} \sum_{\forall f_{i,j}\in s_i} f_{i,j}^{[v_a,v_b]}.\phi \\ &\text{subject to : } \{(7.1),(7.2),(7.3),(7.4),(7.5)OR(7.6)\} \end{aligned} \tag{7.15}$$

Sparse Schedule

In order to increase the porosity of the ST schedules on each link, we propose sparse objective function by Eq. (7.16). The sparse objective function adjusts the ST offsets implicitly by maximizing the sum of slacks between subsequent frames, that are scheduled on the same link. As a result, the ST frames are scheduled in a sparse manner to create unequal slacks for the transmission of the lower priority frames. The sparse objective function is subject to the generic constraints as well as the newly proposed constraints.

$$\begin{aligned} &maximize \sum_{[v_a,v_b]\in L} hopSum^{[v_a,v_b]} \\ &\text{subject to : } \{(7.1),(7.3),(7.4),(7.5)OR(7.6),(7.7),(7.8),(7.10),(7.12)\} \end{aligned} \tag{7.16}$$

Evenly Sparse Schedule

The proposed sparse schedule objective function can be manipulated by an additional constraint, in Eq. (7.17), to create evenly distributed ST frames, or in other words equally-sized slacks on the links.

$$\begin{aligned} &maximize \sum_{[v_a,v_b]\in L} hopSum^{[v_a,v_b]} \\ &\text{subject to}: \{(7.1),(7.3),(7.4),(7.5)OR(7.6),(7.7),(7.8),(7.10),(7.12),(7.13)\} \end{aligned} \tag{7.17}$$

In crux, the minimization and maximization objective functions send ST frames with less number of gate state change, by packing the frames in shared transmission slots. On the other hand, these constraints do not consider slacks between ST frames, thus they cause long queuing times for BE frames either in the beginning or at the end of the hyper period. The sparse and evenly sparse objective functions take advantage of new optimization variables that define slacks after each ST frame on a link. This causes porosity in bandwidth used by the ST frames on each link. Consequently, the BE frames can be fitted in adequate slacks created along the scheduled times for ST frames. The sparse and even sparse objective functions also provide the freedom to squeeze the ST schedule search space by constraining the size of slacks, which is simultaneously beneficial to reduce the optimization time and control the distribution pattern of the slacks.

7.5 Experimental Evaluation

In this section, we present the evaluation setup followed by a simulation-based experimental evaluation to show the effects of proposed objective functions on the QoS of the BE frames.

7.5.1 Evaluation Setup

In order to perform the evaluation, we consider a multi-hop network topology that consists of six end stations. The end stations are connected via two TSN switches as shown in Figure 7.9.

To evaluate the QoS of BE frames we use the NeSTiNg TSN simulation tool [31], which is based on the OMNeT++/INET framework. NeSTiNg facilitates the simulation of networks that are based on the TSN standards. It supports all traffic classes and the TAS mechanism. We developed a plug-in[3] for the NeSTiNg simulation tool that facilitates the insertion of streams, configurations of the gates' states setting and extraction of the simulation results.

[3]https://gitlab.com/Scipsybee/automated-tsn-configuration-plugin

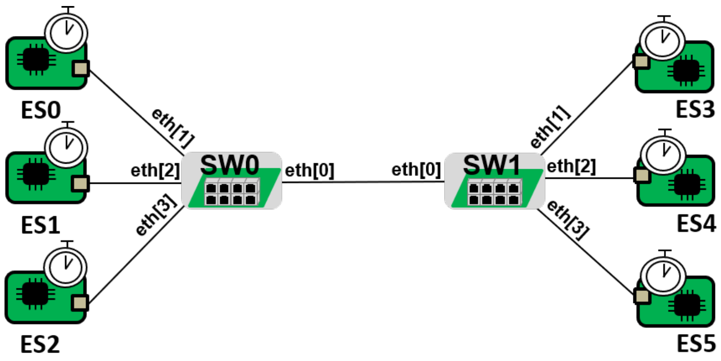


Figure 7.9. Evaluation setup.

In this experiment, we generate random sets of streams, where each stream contains one frame with the maximum size of 1542 Bytes (including the frame header). The source and destination end stations of the streams are selected randomly among the stations shown in the network topology (Figure 7.9). The streams' periods are selected randomly from the set [200, 500, 1000, 2000, 5000] μs to generate a realistic data set according to the real-world automotive benchmarks [32]. We assume that the network speed is 1 $Gbit/s$ in this example. Using the network speed, the transmission time of each frame with 1542 Bytes of frame size is equal to 12.336 μs. The link macrotick for all links is assumed to be 1 μs and we neglect the link propagation delay. The generated streams are selected among ST and BE classes. Three different scenarios are considered in this experiment, where 10 sets of streams are randomly generated in each scenario. Each set contains a mix of BE and ST streams. The three scenarios are as follows:

- Scenario (1) : 10 random streams with 0.5 probability of ST streams and 0.5 probability of BE streams.
- Scenario (2) : 10 random streams with 0.6 probability of ST streams and 0.4 probability of BE streams.
- Scenario (3) : 10 random streams with 0.8 probability of ST streams and 0.2 probability of BE streams.

To schedule the ST streams we consider the four proposed optimization objectives including: (i) minimization; (ii) maximization; (iii) sparse; and (iv) evenly sparse schedule; In the case of sparse and evenly sparse objective functions, $minPorosity$ is assumed to be zero to make the schedules as flexible as possible. Note that increasing the $minPorosity$ value can reduce the number of feasible schedules, while at the same time it can produce larger slacks for transmission of BE frames. The network is simulated according to the schedules generated by the four different objective functions. The two main metrics, measured during the simulations, include the maximum end-to-end delays and deadline misses of the BE frames.

The objective functions and constraints are implemented in Python and solved by Z3's [33] OMT optimization module. The optimizations and simulations were run on an HP Elite Book 820 running Ubuntu OS 18.04.4 LTS with CPU Core i5, 4 $\times$ 2.20 GHz Cores and of 16 GB RAM.

7.5.2 Results discussion

The maximum end-to-end delay of the generated BE streams are measured during the simulation given that the ST streams are scheduled according to the schedules generated by the proposed objective functions. Figure 7.10 illustrates the measured maximum end-to-end delays of the BE streams for the three scenarios, i.e, Scenario (1) in Figure 7.10a, Scenario (2) in Figure 7.10b, and Scenario (3) in Figure 7.10c.

Each bar in the graph shows the measured maximum end-to-end delays of all BE streams in the 10 generated sets by highlighting the minimum, average and maximum of these values. For instance, in Scenario (1) 10 sets of streams were generated randomly where each set consists of 10 streams. The chance of the streams becoming ST or BE were 50% in this scenario. Therefore, we have 10 sets with possibly 5 streams per set belong to the BE class. Consequently, the Sparse bar in Figure 7.10a, for example, shows the maximum end-to-end delays of possibly 50 BE streams, where the average, maximum and minimum values of the measured maximum end-to-end delays among these 50 BE streams are 50.6μs, 98.3μs and 24.5μs, respectively.

As it can be seen in the figure, the minimization objective function gives the worst results compared to the other objective functions with maximum value of 101.2μs in Scenario (1), 102.5μs in Scenario (2) and 90.2μs in Scenario (3). Note that all previous works on the ST scheduling commonly apply the minimization objective function. In the first glance, it can be seen that the maximization objective function outperforms the other objective functions. The main reason is that when the ST schedules are using the time-slots close to their deadlines, more spaces will be available for transmission of possible BE frames. This is in contrast to the minimization objective function that fills the time-slots in the beginning of the ST schedules, leaving spaces for the BE streams after the transmission of ST streams that can potentially result in larger end-to-end delays for the BE frames. However, there are also few extreme cases in which the measured maximum end-to-end delays of BE streams are very high when using the maximization objective function. This is specifically the case in Scenario (2) and Scenario (3). According to the results, the maximization objective function in Scenario (2) resulted in a maximum delay for one BE stream up to 112.2μs and in Scenario (3)

Table 7.1. Number of deadline misses.

	Objective functions	Total deadline misses
Scenario (1)	Maximization	0
	Minimization	2
	Evenly sparse	1
	Sparse	1
Scenario (2)	Maximization	99
	Minimization	102
	Evenly sparse	2
	Sparse	1
Scenario (3)	Maximization	0
	Minimization	0
	Evenly sparse	0
	Sparse	0

up to $212.2\mu s$. These large end-to-end delays are not seen in the sparse and evenly sparse objective functions. The main reason to have these cases with the maximization objective function is that, similar to the minimization objective function, it can pack the ST frames together and prevent transmission of any BE frame for a long duration of time depending on the number of ST frames with close periods. However, the sparse and evenly sparse objective functions can generate porosity in the ST schedules that can be sufficiently used for the BE frames' transmission preventing the long blocking time by the ST frames. Therefore, although on average, the maximization objective function performs better than the rest, the sparse and evenly sparse objective functions have their own benefits in various scenarios, e.g., the one discussed above.

We also measured the deadline misses of BE streams in the generated scenarios. We consider implicit deadlines, i.e., the deadline for each BE stream is assumed to be equal to its period. Table 7.1 shows the total deadline misses among all generated BE streams within each scenario and with each objective function. It can be clearly observed that the minimization objective function can produce more deadline misses compared to the other objective functions, while the sparse and evenly sparse objective functions resulted in fewer deadline misses for the BE streams. For instance, in Scenario (2) the number of deadline misses with minimization and maximization objective functions are 102 and 99, respectively, whereas it is 1 and 2 for sparse and evenly sparse objective functions, respectively. Scenario (3) shows no deadline misses, which can be because of less number of generated BE streams, i.e., 20% chance for the generated streams to be in BE class.

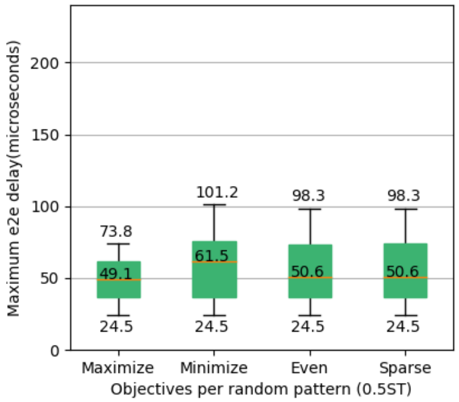
Although the time that it takes to find a feasible and optimized schedule depends on the implementation and the utilized solver, we measured this time in all the experiments. As mentioned before, we used Z3 SMT/OMT solver for this purpose. The average times for each scenario and objective function are shown in Figure 7.11. An interesting observation is that both sparse and evenly sparse objective functions are much faster to give an optimized ST schedule compared to both minimization and maximization objective functions. For instance, in Scenario (3) the amount of time that it takes to deliver an ST schedule using the sparse objective function in average was 4.75 seconds. For the same scenario, the amount of time that it takes to deliver an ST schedule using the minimization and maximization objective functions were 314.34 seconds and 1153.52 seconds, respectively.

7.6 Conclusion and Future Works

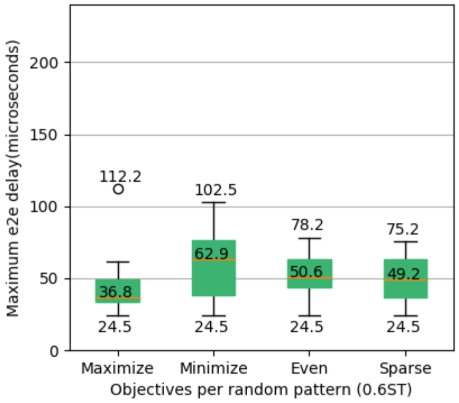
In this paper, we proposed a set of constraints and objective functions to schedule ST traffic in TSN networks considering the QoS of the BE traffic. We argued that in many industrial applications one of the main challenges is to map the legacy traffic into the TSN traffic classes and often the soft real-time traffic are mapped into the BE class. Therefore, a required level of QoS for the BE traffic should be obtained while ensuring a feasible schedule for the hard real-time ST traffic. The solutions proposed in this paper generate feasible schedules for the ST traffic and at the same time significantly reduce the end-to-end delays and the number of deadline misses for the BE traffic. Using a set of experiments, based on the NeSTiNg TSN simulation tool, we showed that the proposed objective functions outperform the state-of-the-art ST scheduling solutions that focus on minimizing the offsets of the ST traffic. We also showed that the feasible ST schedules can be obtained much faster with the proposed solution compared to the state-of-the-art solutions. Although the main focus of the solutions proposed in this paper was on the QoS of BE traffic, we believe that it can also affect positively on the performance of classes A and B. However, a deeper investigation remains for the future as the NeSTiNg simulation tool currently does not support the TAS and CBS mechanisms at the same time. We also plan to investigate the effect of enabling preemption on the quality of service of the BE traffic. Another future research direction is to develop schedulability analysis for the BE traffic to analytically show the performance of the proposed solutions.

The work in this paper is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) through the projects DESTINE and PROVIDENT, and by the Swedish Knowledge Foundation (KKS) through the projects DPAC, HERO & FIESTA. The authors thank all industrial partners, especially Arcticus Systems, Volvo

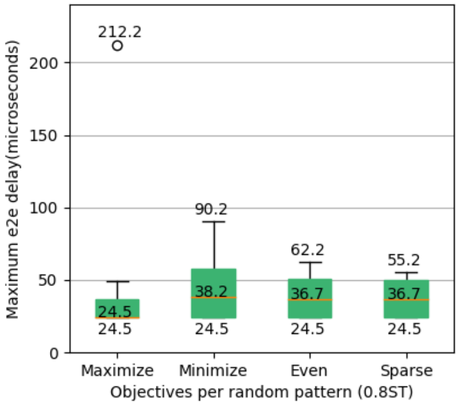
CE, and HIAB Sweden.



(a) Scenario (1).

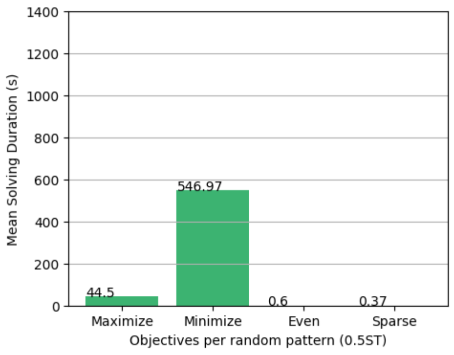


(b) Scenario (2).

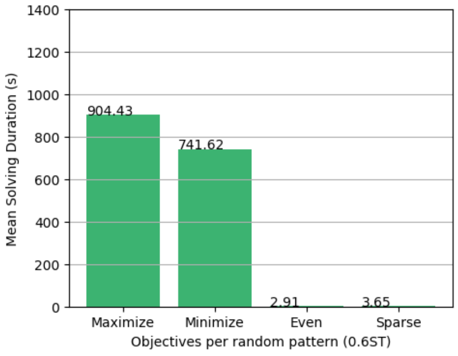


(c) Scenario (3).

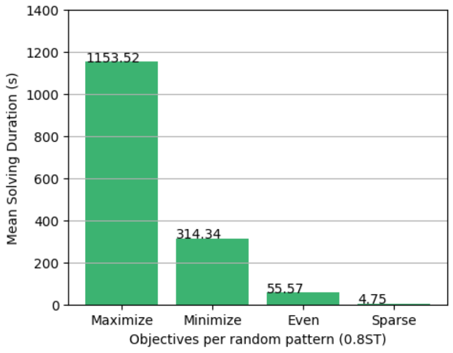
Figure 7.10. Maximum end-to-end delays of the BE streams under various objective functions per traffic distribution case.



(a) Scenario (1).



(b) Scenario (2).



(c) Scenario (3).

Figure 7.11. Duration of schedule synthesis for each distribution pattern per objective function.

Bibliography

[1] IEEE. IEEE Std. 802.1AS, IEEE standard for local and metropolitan area networks-timing and synchronization for time-sensitive applications. 2020.

[2] IEEE. IEEE Std. 802.1Q, IEEE standard for local and metropolitan area networks, bridges and bridged networks. 2018.

[3] L. Lo Bello, R. Mariani, S. Mubeen, and S. Saponara. Recent advances and trends in on-board embedded and networked automotive systems. *IEEE Transactions on Industrial Informatics*, 15(2), 2019.

[4] L. Lo Bello and W. Steiner. A perspective on ieee time-sensitive networking for industrial communication and automation systems. *Proceedings of the IEEE*, 2019.

[5] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner. Scheduling real-time communication in ieee 802.1qbv time sensitive networks. In *Proceedings of the 24th Intl. Conference on Real-Time Networks and Systems*, 2016.

[6] N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop. Window-based schedule synthesis for industrial ieee 802.1qbv tsn networks. In *2020 16th IEEE Intl. Conference on Factory Communication Systems*, 2020.

[7] F. Pozo, G. Rodriguez-Navas, and H. Hansson. Schedule reparability: Enhancing time-triggered network recovery upon link failures. In *2018 IEEE 24th Intl. Conference on Embedded and Real-Time Computing Systems and Applications*, 2018.

[8] M. Pahlevan and R. Obermaisser. Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks. In *2018 IEEE 23rd Intl. Conference on Emerging Technologies and Factory Automation*, Sep. 2018.

[9] V. Gavriluţ, L. Zhao, M. L. Raagaard, and P. Pop. Avb-aware routing and scheduling of time-triggered traffic for tsn. *IEEE Access*, 2018.

[10] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νz - an optimizing smt solver. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*. Springer Berlin Heidelberg, 2015.

[11] Wilfried Steiner. An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks. In *31st IEEE Real-Time Systems Symposium*, 2010.

[12] S. S. Craciunas and R. S. Oliver. Combined task-and network-level scheduling for distributed time-triggered systems. Springer, 2016.

[13] S. S Craciunas, R. S. Oliver, and Wilfried Steiner. Formal scheduling constraints for time-sensitive networks. *arXiv preprint arXiv:1712.02246*, 2017.

[14] S. S. Craciunas and R. S. Oliver. An Overview of Scheduling Mechanisms for Time-sensitive Networks. 2017.

[15] S. S. Craciunas, R. S. Oliver, and W. Steiner. Demo abstract: Slate xns–an online management tool for deterministic tsn networks. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium*, 2018.

[16] R. S. Oliver, S. S. Craciunas, and W. Steiner. Ieee 802.1qbv gate control list synthesis using array theory encoding. In *2018 IEEE Real-Time and Embedded Technology and Applications Symposium*, 2018.

[17] TSNSched. Automated schedule generation for TSN networks, 2020.

[18] A. C. T. dos Santos, B. Schneider, and V. Nigam. Tsnsched: Automated schedule generation for time sensitive networking. In *2019 Formal Methods in Computer Aided Design*. IEEE, 2019.

[19] B. Schneider. Automatic network configuration for real-time, distributed industrial automation systems. In *2019 ACM/IEEE 22nd Intl. Conference on Model Driven Engineering Languages and Systems Companion*, Sep. 2019.

[20] W. Steiner. Synthesis of static communication schedules for mixed-criticality systems. In *2011 14th IEEE Intl. Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, 2011.

[21] L. Zhao, P. Pop, and S. S. Craciunas. Worst-case latency analysis for ieee 802.1qbv time sensitive networks using network calculus. *IEEE Access*, 2018.

[22] Y. Mi, J. Qu, J. Zhang, and M. Yao. A scheduling algorithm of maximize the number of porosity for the time-triggered dima system. In *2020 IEEE 3rd Intl. Conference on Electronics Technology*, May 2020.

[23] M. H. Farzaneh and A. C. Knoll. An ontology-based plug-and-play approach for in-vehicle time-sensitive networking (tsn). In *7th IEEE Annual Information Technology, Electronics and Mobile Communication Conference*, Oct 2016.

[24] M. H. Farzaneh, S. Shafaei, and A. C. Knoll. Formally verifiable modeling of in-vehicle time-sensitive networks (tsn) based on logic programming. In *2016 IEEE Vehicular Networking Conference*, 2016.

[25] M. H. Farzaneh and A. C. Knoll. Time-sensitive networking (tsn): An experimental setup. In *IEEE Vehicular Networking Conference*, Nov 2017.

[26] M. H. Farzaneh, S. Kugele, and A. C. Knoll. A graphical modeling tool supporting automated schedule synthesis for time-sensitive networking. In *2017 22nd IEEE Intl. Conference on Emerging Technologies and Factory Automation*, 2017.

[27] M. H. Farzaneh. *A Modeling Framework to Facilitate Schedule Synthesis of Time-Sensitive Networking*. PhD thesis, Technische Universität München, 2019.

[28] M. Pahlevan, N. Tabassam, and R. Obermaisser. Heuristic list scheduler for time triggered traffic in time sensitive networks. *SIGBED Rev.*, February 2019.

[29] M. Pahlevan, J. Schmeck, and R. Obermaisser. Evaluation of tsn dynamic configuration model for safety-critical applications. In *2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, pages 566–571, 2019.

[30] F. Dürr and N. Nayak. No-wait packet scheduling for ieee time-sensitive networks (tsn). In *Proceedings of the 24th Intl. Conference on Real-Time Networks and Systems*, 2016.

[31] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, and K. Rothermel. NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++. In *Proceedings of the 2019 Intl. Conference on Networked Systems*. IEEE, March 2019.

[32] S. Kramer, D. Ziegenbein, and A. Hamann. Real World Automotive Benchmarks for Free. In *6th Intl. Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems*, 2015.

[33] Z3Py. Z3 is a theorem prover from microsoft research, 2020.

Chapter 8

Paper C: Schedulability Analysis of Best-effort Traffic in TSN Networks

Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Sara Afshar, Saad Mubeen.

In the 26^{th} IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2021.

Abstract

This paper presents a schedulability analysis for the Best-Effort (BE) traffic class within Time Sensitive Networking (TSN) networks. The presented analysis considers several features in the TSN standards, including the Credit-Based Shaper (CBS), the Time-Aware Shaper (TAS) and the frame preemption. Although the BE class in TSN is primarily used for the traffic with no strict timing requirements, some industrial applications prefer to utilize this class for the non-hard real-time traffic instead of classes that use the CBS. The reason mainly lies in the fact that the complexity of TSN configuration becomes significantly high when the time-triggered traffic via the TAS and other classes via the CBS are used altogether. We demonstrate the applicability of the presented analysis on a vehicular application use case. We show that a network designer can get information on the schedulability of the BE traffic, based on which the network configuration can be further refined with respect to the application requirements.

8.1 Introduction

Today's vehicular embedded systems deal with many challenges, such as scalability in data communication, computational complexity and guaranteeing determinism for hard real-time traffic. These challenges are due to increased functionality of automotive systems, which demands coexistence of diverse applications within the same network. Diversity of applications can be in terms of priority, timing constraints or bandwidth constraints. Therefore in recent years, IEEE Time-Sensitive Networking (TSN) task group has been formed to provide features that can support such diversity in communication and act as a backbone communication-network for some industrial domains; in particular for the vehicular on-board communication-systems [1, 2].

TSN standards are toolboxes where various features depending on the application can be used to improve the performance of the communication in the application. In particular, TSN standards allow temporal isolation of the Scheduled Traffic (ST), which is sent according to a static schedule created offline. The temporal isolation is realized by the Time-Aware Shaper (TAS) mechanism using gates operation on ports of a TSN switch. This guarantees timing determinism for ST. In addition to the TAS mechanism, TSN standards define the Credit-Based Shaper (CBS) mechanism that allows reservation of the bandwidth by reserving links' capacity based on credits associated to each class. The shapers define different traffic classes in TSN networks, being classes A, B and Best-Effort (BE). Classes A and B use the CBS mechanism, while class BE does not use the CBS mechanism. Class A has a higher priority than class B, and class BE has the lowest priority in the network. Hence, BE traffic will only be sent when no higher priority traffic is contending for the link.

One of the main industrial domains, where using TSN is gaining significant momentum is the vehicular domain due to their advancements in functionalities, smart devices, and high-bandwidth sensors. However, redesigning and replacing the existing communication systems is not reasonable, mainly because of extra costs imposed by the new design. Therefore, the vehicular applications consider the TSN network as a backbone network supporting high-bandwidth between several Electronic Control Units (ECU). One of the challenging and non-trivial tasks is to support the legacy traffic and map them into the TSN traffic classes. Due to the complexity of applying a combination of TSN features, there is a high tendency in vehicular industry to use only the ST class for hard real-time strictly periodic traffic. For example, using a combination of the TAS and CBS mechanisms in practice is not realized easily, due to the complexity of the CBS configuration. Also, it is common to use the BE class for the legacy traffic, that has no strict deadlines but a minimum quality-of-service (QoS) demand. Since, the BE class is simple to use and configure, obtaining a level of QoS via schedulability analysis is useful for a large class of BE traffic.

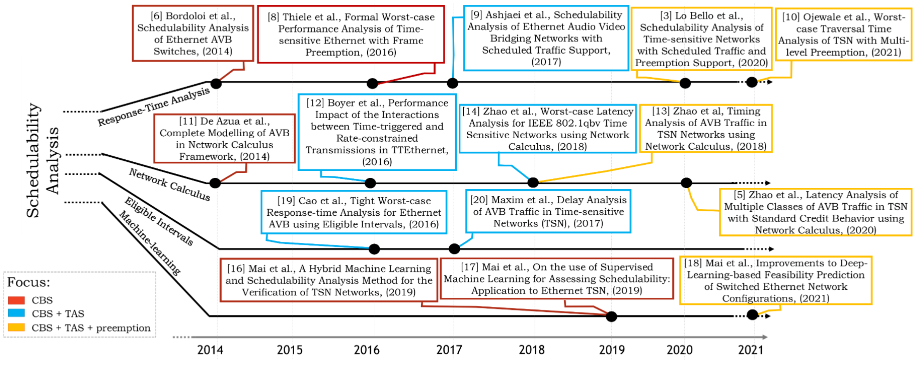

[6] Bordoloi et al., Schedulability Analysis of Ethernet AVB Switches, (2014)
[8] Thiele et al., Formal Worst-case Performance Analysis of Time-sensitive Ethernet with Frame Preemption, (2016)
[9] Ashjaei et al., Schedulability Analysis of Ethernet Audio Video Bridging Networks with Scheduled Traffic Support, (2017)
[3] Lo Bello et al., Schedulability Analysis of Time-sensitive Networks with Scheduled Traffic and Preemption Support, (2020)
[10] Ojewale et al., Worst-case Traversal Time Analysis of TSN with Multi-level Preemption, (2021)
Schedulability Analysis
Response-Time Analysis
[11] De Azua et al., Complete Modelling of AVB in Network Calculus Framework, (2014)
[12] Boyer et al., Performance Impact of the Interactions between Time-triggered and Rate-constrained Transmissions in TTEthernet, (2016)
[14] Zhao et al., Worst-case Latency Analysis for IEEE 802.1qbv Time Sensitive Networks using Network Calculus, (2018)
[13] Zhao et al, Timing Analysis of AVB Traffic in TSN Networks using Network Calculus, (2018)
Network Calculus
Eligible Intervals
Machine-Learning
[19] Cao et al., Tight Worst-case Response-time Analysis for Ethernet AVB using Eligible Intervals, (2016)
[20] Maxim et al., Delay Analysis of AVB Traffic in Time-sensitive Networks (TSN), (2017)
[5] Zhao et al., Latency Analysis of Multiple Classes of AVB Traffic in TSN with Standard Credit Behavior using Network Calculus, (2020)
[16] Mai et al., A Hybrid Machine Learning and Schedulability Analysis Method for the Verification of TSN Networks, (2019)
[17] Mai et al., On the use of Supervised Machine Learning for Assessing Schedulability: Application to Ethernet TSN, (2019)
[18] Mai et al., Improvements to Deep-Learning-based Feasibility Prediction of Switched Ethernet Network Configurations, (2021)
Focus:
🟥 CBS
🟦 CBS + TAS
🟨 CBS + TAS + preemption
2014
2015
2016
2017
2018
2019
2020
2021

Figure 8.1. Timeline of schedulability analysis techniques for TSN since 2014.

To this end, none of the existing works support the schedulability analysis of the BE class, when the TAS, CBS and frame preemption are used in the network. The main contribution in this paper is to develop a schedulability analysis that can verify the worst-case response time of each individual BE message in the network when CBS, TAS and frame preemption are used. Furthermore, the paper shows the applicability of the analysis on a use case from the vehicular domain.

8.2 Background and Related Work

In general, there are four main analysis techniques for TSN networks: (1) RTA, (2) network calculus, (3) use of eligible intervals, and (4) use of machine learning. Figure 8.1 shows a timeline of the existing schedulability analysis approaches for TSN. According to [2], the focus of the first works in the existing schedulability analysis has evolved from only supporting CBS to more sophisticated models, which include the combination of CBS with time-triggered traffic and frame preemption support. There are several recent works that provide schedulability analysis of TSN networks considering the frame preemption [3], [4, 5]. Among the above mentioned techniques, this paper focuses on the worst-case RTA. More specifically, we extend the work in [3] to provide schedulability analysis for BE class in TSN networks. The following sections present a summary of the most significant research in the area of schedulability analysis for TSN.

8.2.1 Response Time Analysis (RTA) for TSN

The work in [6] is one of the first works aiming at the utilization of AVB for in-vehicle communication. The proposed RTA only supports CBS and the class A traffic is assumed to be the highest-priority traffic. Hence, the work in [6] does not consider any interference by the ST class. The RTA was further developed in several works [7] to support the hard real-time time-triggered traffic in TSN, which needs to be shaped by deterministic traffic shaper mechanisms, such as the TAS or the peristaltic shaper. A comparative evaluation of these two shapers discussed in [7] shows that shaping the hard real-time traffic based on the peristaltic shaper can cause larger blocking times to the message under analysis. Hence, the work in [8] extended the previous work to provide the worst-case RTA for TSN supporting the TAS and preemption. The work in [9] considered the CBS and TAS mechanisms in combination, where the TAS mechanism was a variation of the TSN standard. Furthermore, the work in [3] proposed the worst-case RTA that considers the TAS, CBS and different variations of frame preemption support (i.e., with and without Hold and Release mechanisms according to the standard). Finally, the work in [10] introduced a worst-case traversal analysis for TSN, which is based on a system model that allows multiple preemption levels [4].

8.2.2 Analysis for TSN based on Network Calculus

Network calculus is a well-known technique to calculate the worst-case delays in networks. A network calculus analysis for AVB was introduced in [11]. Later, the work in [12] provided an approach to integrate the timing analysis for the periodic time-triggered traffic and the rate-constrained sporadic traffic in TTEthernet. Though TTEthernet has several similarities to TSN, it does not feature the CBS mechanism.

Within the context of TSN, the works in [13] and [14] considered the influence of the scheduled traffic on the AVB traffic in preemption mode. The main focus of this work is to employ network calculus to find tighter worst-case delay bounds for classes A and B. The work was further developed in [5] to support multiple AVB classes in the system model and to include the effects of traffic classes such as ST, A and B. Finally, the work in [15] presents the state of the art in network calculus-based TSN methods.

8.2.3 Analysis for TSN based on Machine Learning

An interesting approach for verification of TSN networks and their schedulability analysis using machine learning techniques was followed in the works presented in [16] and [17]. This approach mainly focuses on verifying the feasibility of TSN configurations by combining a schedulability analysis and a machine learning technique. The work in [18] further improved the approach by applying deep learning-based techniques. However, these techniques are not primarily proposed to provide predictability guarantees for TSN networks, yet are interesting for verifying the TSN network configurations.

8.2.4 Analysis for TSN based on Eligible Intervals

Besides the aforementioned works, a technique based on the concept of eligible intervals was proposed in the context of AVB network considering solely the CBS mechanism in [19]. Further, the work in [20] used the same concept to analyze the delays of classes A and B in the presence of the ST class.

To sum up, the previous schedulability analysis techniques have addressed several features in TSN, but are still limited to the analysis of classes A and B. This seems intuitive as the BE class was not supposed to be used for traffic with timing requirements. However, in order to avoid the complexity of the CBS traffic, the designers of TSN applications often prefer to assign the legacy traffic with timing requirements to the BE class instead of the CBS traffic (classes A and B). Hence, schedulability analysis of the BE traffic class is needed. Provisioning of the RTA for the BE traffic class in TSN networks is the main focus of this paper.

8.3 System Model

The system model consists of two parts. First, the network model defines the physical attributes of the network. Second, the traffic model describes the message attributes that are communicated between the source and destination end stations.

8.3.1 Network Model

In the network model, we assume TSN switches that support CBS, TAS, and clock synchronization. The network topology is modelled through a set of links shown by $\mathcal{L}$ that contains all network links. A typical link in the network is indicated by l, where $l \in \mathcal{L}$. Links are two-directional connections linking an end station to a switch and one switch to another switch. R specifies the network bandwidth per link, i.e., the maximum allowed load on the links. We assume that all the links have the same bandwidth. Moreover, the parameters $\alpha^{+}_{z,l}$ and $\alpha^{-}_{z,l}$ are the idleSlope and sendSlope values associated with the link l, which are only applicable when traffic from credit-based class z uses the link. The idleSlope is the rate of increasing the credit for a class of traffic when there is a pending message for transmission in the class. The sendSlope is the rate of credit consumption when the message is being transmitted. The sum of idleSlope and sendSlope is equal to R. Finally, the delay due to hardware design of the switch is assumed to be bounded by γ.

8.3.2 Traffic Model

The traffic is modelled by a message, m_i. A message contains a stream of data to be transmitted from the source end station to the destination end station. An instance of a message is called a frame. We assume that traffic preemption is enabled. The traffic class ST is configured as an express class, while other classes are preemptable (not express). Therefore, ST messages can preempt messages from the other classes, but messages from the other classes cannot preempt the lower-priority messages. The fragments of a preempted message obtain an individual header; therefore, preemptions cause extra overhead due to the extra transmission time required for the headers of the fragments. The parameter v is the transmission time of an Ethernet frame header. The attributes of the message are represented by the tuple $\langle C_i, T_i, D_i, P_i, \mathcal{O}_i^l, \mathcal{L}_i \rangle$. Where C_i is the transmission time of m_i. The transmission time is calculated based on R and the size of the message. A guard band is a reserved slot added before the ST schedule slots in case the preemption by the ST class is enabled to prevent any interference by the lower-priority messages that cannot be preempted. The duration of the guard band is represented by λ. According to the IEEE 802.3br standard, a message fragment less than 123 $Bytes$ cannot be preempted. T_i is the message m_i's period and D_i specifies the end-to-end deadline of m_i. We consider implicit deadlines, i.e., the deadline of each message is assumed to be equal to its period. The message priority is represented by P_i. We assume that the possible values of P_i can be assigned from the set of class IDs in the set $\{ST, A, B, BE\}$, and each TSN class is associated with a queue. Moreover we consider only two AVB classes, but it's worth noting that up to 8 AVB classes can be supported by the shaper mechanism. In case the message is from the ST class, deterministic schedules on each of the links need to be specified at the Gate Control List (GCL) of the TSN switches. Therefore, the parameter $\mathcal{O}_i^l$ represents the set of offsets of the ST message m_i on link l within $\mathcal{L}_i$, where $\mathcal{L}_i$ is the set of links in the route of m_i from the source end station to the destination end station, and $\mathcal{L}_i \in \mathcal{L}$. We assume that the ST schedules (i.e., the offsets for the ST messages) are given using a scheduling technique, e.g., [21]. J_i^l is the jitter on the link l, which is the delay variations (bound) in the transmission of the message m_i on the link l.

8.4 Revisiting RTA for TSN without the BE Traffic

The worst-case response time of a message consists of three components: (1) interference from the messages belonging to the higher-priority traffic (hp), (2) interference from the messages belonging to the same priority traffic (sp), and (3) blocking from the messages belonging to the lower-priority traffic (lp). The RTA for BE traffic in TSN, presented in this paper, is built upon the existing RTA [3], which is revisited in this section. The following subsections present the worst-case RTA for the messages belonging to classes A and B.

8.4.1 RTA for Class A Messages

In order to explain various factors contributing to the response time of a class A message, the transmission trace of an example is presented in Figure 8.2. A message could be routed through several links and switches; hence the worst-case RTA must be performed per link. The trace in Figure 8.2 shows a shared link between messages from three different priority classes in TSN, namely classes ST, A, and B. For example, there are four periodic frames of an ST message that are activated at every 5 time units. The transmission time of the ST message is 1 time unit. Class A contains a message A_1 with the period of 20 time units and transmission time of 3 time units. The period and transmission time of the class B message are 20 and 3 time units respectively. Figure 8.2 shows the transmission trace until the hyperperiod (20 time units) of all the messages. Moreover, the upward arrows indicate the activation times of the messages in the hyperpeiod.

We are interested in the response time of message A_1, where we need to find a critical instant candidate which can lead to the worst-case scenario. The worst-case scenario for A_1 occurs when the higher and the same priority messages are activated at the same time as the activation of A_1, i.e., at time 0 in Figure 8.2. For simplicity of the trace, we excluded the same priority interference, and the credits of classes A and B are not shown. However, we assumed that the credits are 0 at the time of the message's activation. Moreover, the worst-case scenario also considers that there is an ongoing transmission of a lower-priority message (B_1) on the port when the message under analysis is activated. Since the higher priority class with respect to class A is set to express, these express messages are handled by the TAS allowing the express messages to preempt the lower-priority messages. In this case, preemption overhead due to adding an Ethernet frame header to each preempted fragment of the messages needs to be taken into account. This interference on A_1 can be seen at time 5 in Figure 8.2.

Another component of the worst-case scenario is when the lower-priority message from class B is activated at a slightly earlier time than the message under analysis from

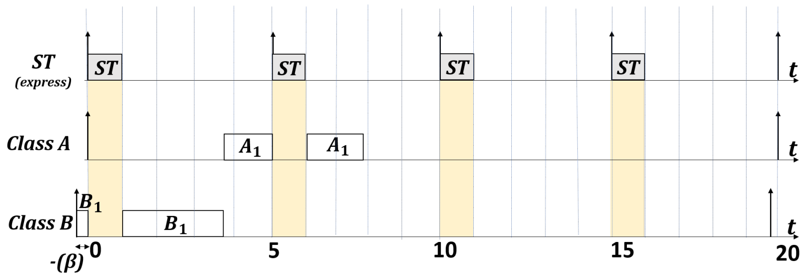


Figure 8.2. Trace of a message set: class A message under analysis.

class A, at time $-\beta$ in Figure 8.2. The message B_1 starts its transmission at $-\beta$ as there are no higher priority messages that are queued for transmission at the same time. Although at time 0 the frame A_1 is activated, it cannot be preempted by B_1 because it is set to preemptable. However, the ST frames can preempt both B_1 and A_1, as can be seen in Figure 8.2. After full transmission of B_1, the frame A_1 can be transmitted, although it may still be preempted by the ST frames.

In summary, the worst-case response time for the class A message consists of: (i) interference from the messages belonging to the ST class; (ii) ST preemption overhead; (iii) blocking due to lower-priority messages belonging to the B and BE classes; and (iv) interference from other same priority messages from class A. The following subsections explain each of the elements contributing to the worst-case response time of a class A message.

Interference from higher-priority messages

Since the ST class is scheduled offline, an offset per link is assigned to each ST message. In order to consider the offsets of the ST messages in the analysis, the offset-based analysis in [22] is adapted in which a transaction is defined to contain several tasks. The adaptation of the transactional model into the ST messages is as follows. Since in the system model, we assumed one ST class using one queue of each port, then all ST messages form one transaction and its period is the least common multiple (LCM) of all ST periods. Figure 8.3 represents an example of two ST messages in a transaction, namely message x and message w, with the periods T_x and T_w, respectively. The transaction period is indicated by T, which is the LCM of T_x and T_w. The parameter k represents an instance of a message within the transaction's period. That is, $k = [1, n]$, where n is equal to the total number of instances of the message within the transaction's period. For example, n for the message w within T is 2 as there are two frames of message w during T. Similarly, n for x within T equals to 1.

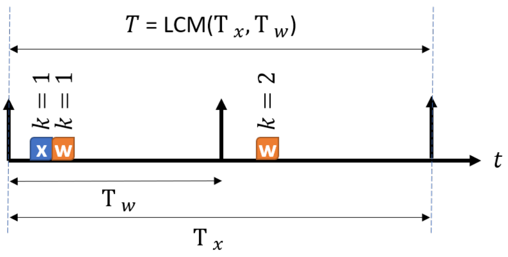


Figure 8.3. An example of a transaction.

Since the activation time of the message under analysis from class A is unknown, each activation of an ST frame within the hyperperiod can be a critical-instant candidate for the class A message. The largest response time caused by the ST class interference to the message under analysis can be found by generating critical-instant candidate combinations from the ST messages. Therefore, it is assumed that the message under analysis is activated at time 0. Then, the phase of the ST transaction is shifted such that the activation of one of the ST messages coincides with the activation of the message under analysis at each critical-instant combination. Since multiple frames of a message can appear in the hyperperiod, i.e., multiple frames in the ST transaction (Figure 8.3), the phase should be derived for all frames within the transaction. In order to derive the offsets for all frames in the transaction, we use Eq. (8.1), where I_j^l is an array containing the offsets of all frames of ST message m_j on link l during the transaction period T_{lcm} (hyperperiod of all ST messages).

$$I_j^l = \{(k-1)T_j + O_j^l - \lambda : k = 1..n, n = \frac{T_{lcm}}{T_j}\} \tag{8.1}$$

The phase between ST message m_j and the k^{th} frame of the critical-instant candidate ST message m_c on link l is denoted by $\Phi_{jc[k]}^l$, that is calculated by Eq. (8.2).

$$\Phi_{jc[k]}^l = (O_j^l - I_c^l[k]) \ mod \ T_{lcm} \tag{8.2}$$

O_j^l is the offset of ST message m_j on link l, whereas $I_c^l[k]$ is the offset of k^{th} frame of m_c on l. The worst-case interference by ST messages is calculated by Eq. (8.3), when candidate ST message m_c coincides with the critical instant [3].

$$W_{c[k]}^l(t) = \sum_{\forall j \in ST \ \wedge l \in \mathcal{L}_j} \left(\left\lfloor \frac{\Phi_{jc[k]}^l}{T_{lcm}} \right\rfloor + \left\lceil \frac{t - \Phi_{jc[k]}^l}{T_{lcm}} \right\rceil \right) C_j \tag{8.3}$$

As it is mentioned earlier, in case of any preemption, the fragmented frame will again obtain a frame header. Therefore, the frame becomes larger in size with each preemption. The impact of this preemption overhead is calculated by Eq. (8.4).

$$V^l_{c[k]}(t) = \sum_{\forall j \in ST \, \wedge l \in \mathcal{L}_j} \left(\left\lfloor \frac{\Phi^l_{jc[k]}}{T_{lcm}} \right\rfloor + \left\lceil \frac{t - \Phi^l_{jc[k]}}{T_{lcm}} \right\rceil \right) v \tag{8.4}$$

Blocking by lower priority messages

Since the CBS controls the transmission of class A traffic, the blocking of class A message under analysis can occur under two circumstances. Firstly, when the lower-priority messages are activated slightly earlier than the activation of the class A message. Secondly, a lower-priority message takes the port for transmission while the class A message is activated but the credit for class A is negative. It has been shown in [6] that several lower-priority messages can block the class A message. However, if the same priority interference is inflated as in Eq. (8.6) then it is safe to consider only one blocking message from the lower-priority traffic as shown in Eq. (8.5).

$$BB^l_i = \max_{\substack{\forall m_j \in lp(m_i) \\ \wedge \, l \in \mathcal{L}_j}} \{C_j\} \tag{8.5}$$

Interference from same priority messages

To cover the need for accounting the blocking due to the mentioned circumstances, the same priority interference should be inflated by $\left(1 + \frac{\alpha^-_{A,l}}{\alpha^+_{A,l}}\right)$. Therefore, Eq. (8.6) calculates the same priority interference for class A message under analysis m_i. For more information and the proofs please refer to [6] (and subsequently in [3]).

$$ISA^l_i = \sum_{\substack{\forall m_j \in sp(m_i), i \neq j \\ \wedge \, l \in \mathcal{L}_j}} \left(1 + \frac{\alpha^-_{A,l}}{\alpha^+_{A,l}}\right) C_j \tag{8.6}$$

In the above equation, $\alpha^+_{A,l}$ and $\alpha^-_{A,l}$ are the idleSlope and sendSlope values of class A on the link l, respectively.

Response-time calculations

The response-time analysis iteratively considers the blocking and interference on the message under analysis (m_i) on the link l in time intervals (t). The iterative process is continued until the values of the computed response times are equal in two consecutive iterations or the value of the response time exceeds the corresponding deadline. Consequently, in a deadline-constrained model, we call a message schedulable if the response time of the message is less than or equal to the message's deadline. The response time of a message in class A is calculated by Eq. (8.7).

$$RT^{l,(x)}_{ic[k]} = W^l_{c[k]}(RT^{l,(x-1)}_{ic[k]}) + V^l_{c[k]}(RT^{l,(x-1)}_{ic[k]}) + BB^l_i + ISA^l_i + C_i \tag{8.7}$$

Where the index of the current and the previous iterations are specified by (x) and $(x-1)$, respectively. The worst-case response time on link l is the maximum value among all calculated response times based on all critical instant candidates' combinations of m_c, as shown in Eq. (8.8).

$$RT^l_i = \max_{\forall m_c \& \forall k} \{RT^l_{ic[k]}\} \tag{8.8}$$

8.4.2 RTA for Class B Messages

Class B is non-express similar to class A. Hence, the four elements that influence the worst-case response time of a class B message include: (i) interference from the higher-priority ST class and class A messages; (ii) class ST preemption overhead; (iii) blocking by class BE messages; (iv) interference from other same priority messages from class B. Figure 8.4 shows an example with four classes of ST, A, B and BE. In this example, we are interested in one of the critical instant candidates of the class B message, i.e., B_1. Similar to the previous example in Figure 8.2, the worst-case scenario considers that a lower-priority message BE_1 has started slightly before the activation of B_1. Since class B is not express, B_1 cannot preempt the transmission of BE_1. When the transmission of BE_1 is completed, a higher-priority class A message, A_1, is activated. Assuming that the credit for class A is zero or positive, A_1 will be transmitted first. Afterwards, B_1 can be transmitted. As shown in Figure 8.2, the ST messages can preempt all lower-priority messages, including B_1. An interesting observation in this example is that the higher-priority interference on class B messages can be preemptive (via class ST messages) or non-preemptive (via class A messages) at the same time.

Interference from higher-priority messages

Interference from the ST messages as well as the preemption overhead due to the ST messages is calculated using Eq. (8.3) and Eq. (8.4), respectively. The class A traffic is not express in contrast to the ST class. Therefore, class A messages cannot preempt any of the lower-priority messages. As a result, the interference of class A follows a

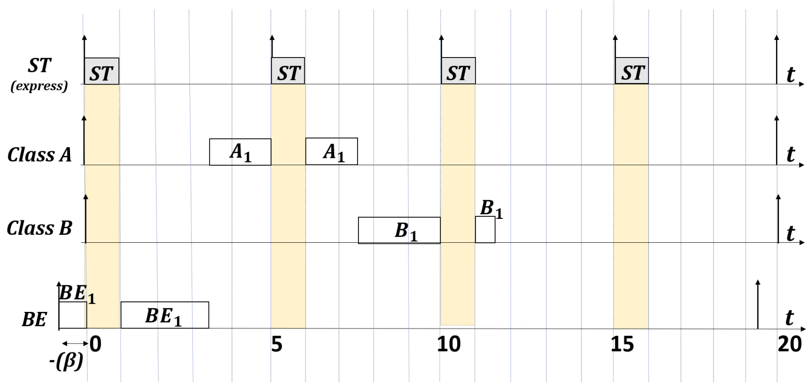


Figure 8.4. Trace of a message set: class B message under analysis.

non-preemptive high-priority interference model. When the link is already occupied by other messages, the current frame of the message under analysis must be queued until the link is idle. Eq. (8.9) calculates the high-priority interference of class A messages on the class B message, where the queuing delay is denoted by $\omega^l_{ic[k]}(q)$.

$$IA^l_i(\omega^l_{ic[k]}(q)) = \sum_{\substack{\forall m_j \in classA, i \neq j \\ \wedge\, l \in \mathcal{L}_j}} \left\lfloor \frac{\omega^l_{ic[k]}(q) + J^l_j}{T_j} + 1 \right\rfloor C_j \tag{8.9}$$

Above, $\omega^l_{ic[k]}(q)$ is the queuing delay on the link l when the k^{th} frame of the critical instant candidate (m_c) coincides with the critical instant. Moreover, it has been shown in [3] that any frame of a message under analysis (m_i) may lead to the worst-case situation (not necessarily the first frame of the message). Thus, q frames should be evaluated for the worst-case analysis. J^l_j is the jitter of each frame of class A message m_j on link l.

Blocking by the lower-priority messages

BE class is the only lower priority traffic than the message under analysis (m_i) from class B. The blocking by this class is presented as BBE^l_i, which is calculated the same as BB^l_i in Eq. (8.5).

Interference from same priority messages

The same priority interference of class B messages on the message under analysis (m_i) is calculated by Eq. (8.10), which takes into account the inflation factor of class B. Besides, the equation considers that the worst-case scenario can be caused by any of the q instances from the message under analysis (m_i). For more information and the proofs please refer to [6] (and subsequently in [3])

$$ISB_i^l = \sum_{\substack{\forall m_j \in sp(m_i), i \neq j \\ \wedge l \in \mathcal{L}_j}} \left(1 + \frac{\alpha_{B,l}^-}{\alpha_{B,l}^+}\right) C_j \left\lfloor \frac{(q-1)T_i}{T_j} + 1 \right\rfloor \tag{8.10}$$

Response-time calculations

The response time calculations of class B are performed in two phases. In the first phase, the queuing delay from all sources of interference and blocking that have an influence on the q^{th} frame of the message under analysis (m_i) is iteratively calculated using Eq. (8.11).

$$\begin{aligned} \omega_{ic[k]}^l(q) = W_{c[k]}^l(\omega_{ic[k]}^l(q)) + V_{c[k]}^l(\omega_{ic[k]}^l(q)) + \\ IA_i^l(\omega_{ic[k]}^l(q)) + BBE_i^l + ISB_i^l + (q-1)C_i \end{aligned} \tag{8.11}$$

In the second phase, the message m_i can only be interfered by ST messages preemptively and not by other classes as their interference is already accounted in the first phase. Therefore, the response time of m_i when m_c from ST class coincides the critical instant for the q^{th} frame is calculated by Eq. (8.12). Note that the second phase starts at time $\omega_{ic[k]}^l(q)$, i.e., after the calculation of the busy period.

$$\begin{aligned} RT_{ic[k]}^{l,(x)}(q) = \omega_{ic[k]}^l(q) + W_{c[k]}^l(RT_{ic[k]}^{l,(x-1)}(q)) + \\ V_{c[k]}^l(RT_{ic[k]}^{l,(x-1)}(q)) + C_i - (q-1)T_i \end{aligned} \tag{8.12}$$

Finally, Eq. (8.13) calculates the maximum value of response time, which is the maximum value of $RT_{ic[k]}^l$ calculated for the q^{th} frame of the message under analysis (m_i).

$$RT_{ic[k]}^l = \max_{q=1..q_{max}} \{RT_{ic[k]}^l(q)\} \tag{8.13}$$

8.5 Proposed RTA for the BE traffic in TSN

This section extends the existing RTA for TSN to support the class BE messages. Class BE does not utilize the express mode. Moreover, the traffic passing from the class BE queue is subjected to interference by the CBS and ST classes. Since the system model does not contain any lower-priority classes than the class BE, unlike other classes, the class BE messages do not experience any blocking delay. Figure 8.5 shows an example of a BE frame's transmission trace and the interference by the higher-priority messages. There is a message belonging to each of these classes. There are four periodic frames of an ST message that are activated every 5 time units. The transmission time of the ST message is 1 time unit. Each message in classes A, B and BE has a period of 20 time units and a transmission time of 3 time units.

A critical instant of the class BE message occurs when it is activated at the same time with the higher-priority messages. Firstly, the class ST message that is in the express mode, preempts the class BE message and its higher-priority messages from classes A and B. The message belonging to class A has subsequently higher priority than the messages of classes B and BE. Therefore, the class A message (in case of positive credit) is transmitted as soon as the link is freed by the class ST message. Similarly, the class B message is transmitted after the transmission of class A message is completed. The class BE message is also fragmented due to the preemption caused by the ST frames. Note that the class BE message is interfered by the class ST messages preemptively and by classes A and B messages non-preemptively. The elements that influence the RTA of class BE frames, include: (i) interference from class ST, classes A and B as the higher-priority classes; (ii) ST preemption overhead; (iii) interference from other same priority messages from class BE.

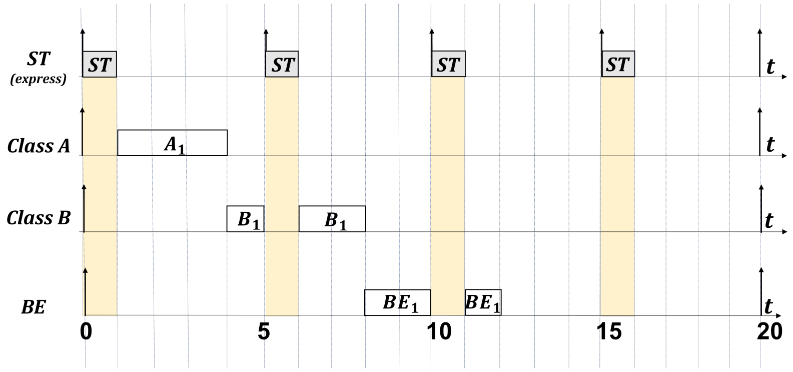


Figure 8.5. Trace of a message set: class BE message under analysis.

8.5.1 Interference from Higher-priority Messages

The set of higher-priority messages with respect to the class BE message includes messages from classes ST, A and B. The class ST messages interfere with the BE message according to the TAS schedules. Whereas, the classes A and B are shaped based on the corresponding class' credit. Consequently, we can expect the same effect of interference and preemption overhead due to the ST schedules as presented for RTA of class A and class B in Section 8.4.1. Eq. (8.14) calculates the interference by classes A and B based on the queuing delays in a similar way to the existing non-preemptive interference.

$$IAB_i^l(\omega_{ic[k]}^l(q)) = \sum_{\substack{\forall m_j \in classA \cup classB, i \neq j \\ \wedge\, l \in \mathcal{L}_j}} \left\lfloor \frac{\omega_{ic[k]}^l(q) + J_j^l}{T_j} + 1 \right\rfloor C_j \tag{8.14}$$

where, J_j^l is the jitter of higher-priority messages from classes A and B that cause the queuing delay to the q^{th} frame of the message under analysis (m_i). The calculation of J_j^l for both classes A and B will be presented at the end of this section.

8.5.2 Interference from the Same Priority Messages

In a deadline-constrained model, only one frame of the same priority message can be queued before the queuing of the message under analysis. Eq. (8.15) calculates the same priority interference for m_i in class BE.

$$ISBE_i^l = \sum_{\substack{\forall m_j \in sp(m_i), i \neq j \\ \wedge l \in \mathcal{L}_j}} C_j \left\lfloor \frac{(q-1)T_i}{T_j} + 1 \right\rfloor \tag{8.15}$$

Note that unlike the same priority interference for classes A and B, the same priority interference for class BE is not inflated. The reason is that the inflation factor is needed to cover the case of multiple blocking by lower-priority messages. As in this model, the BE class is the lowest-priority class, the concept of inflation factor becomes irrelevant.

8.5.3 Response-time Calculations

The RTA for class BE messages is performed in two phases similar to the analysis for class B (in Section 8.4.2). Therefore, in the first phase of the analysis, the queuing delay from all sources of interference that have an influence on the q^{th} frame of the message under analysis (m_i) from class BE is calculated by Eq. (8.16).

$$\begin{gathered}\omega^l_{ic[k]}(q) = W^l_{c[k]}(\omega^l_{ic[k]}(q)) + V^l_{c[k]}(\omega^l_{ic[k]}(q)) + \\ IAB^l_i(\omega^l_{ic[k]}(q)) + ISBE^l_i + (q-1)C_i\end{gathered} \tag{8.16}$$

where, the queuing delay $\omega^l_{ic[k]}(q)$ includes: (i) preemptions by the ST class $W^l_{c[k]}(\omega^l_{ic[k]}(q))$; (ii) ST preemption overhead $V^l_{c[k]}(\omega^l_{ic[k]}(q))$; (iii) credit-shaped high-priority traffic interference by classes A and B ($IAB^l_i(\omega^l_{ic[k]}(q))$); and (iv) the transmission time of all the q frames of the message under analysis that are waiting in the queue.

In the second phase, the queuing delay of the BE message under analysis which is derived in the previous stage is utilized to compute the response time of the message under analysis (m_i). In this phase, m_i can only be interfered by the ST messages preemptively. The response-time calculations in the second phase are presented by Eq. (8.17).

$$\begin{gathered}RT^{l,(x)}_{ic[k]}(q) = \omega^l_{ic[k]}(q) + W^l_{c[k]}(RT^{l,(x-1)}_{ic[k]}(q)) + \\ V^l_{c[k]}(RT^{l,(x-1)}_{ic[k]}(q)) + C_i - (q-1)T_i\end{gathered} \tag{8.17}$$

The worst-case response time of m_i from class BE is the maximum $RT^l_{ic[k]}$ of the k^{th} critical instant candidate combination (m_c) as shown in Eq. (8.18).

$$RT^l_{ic[k]} = \max_{q=1..q_{max}} \{RT^l_{ic[k]}(q)\} \tag{8.18}$$

Where q_{max} represents the maximum number of frames of the message under analysis that are queued for transmission during its maximum busy period. The value of q_{max} is derived as the smallest positive integer value that satisfies the inequality (8.19).

$$\begin{gathered}W^l_{c[k]}(\omega^l_{ic[k]}(q)) + V^l_{c[k]}(\omega^l_{ic[k]}(q)) + IAB^l_i + qC_i + \\ \sum_{\substack{\forall m_j \in classA \cup classB, i \neq j \\ \wedge\ l \in \mathcal{L}_j}} \left\lceil \frac{\omega^l_{ic[k]}(q) + J^l_j}{T_j} \right\rceil C_j \leq q.T_i\end{gathered} \tag{8.19}$$

The worst-case response time of the class BE message (m_i) on link l is equal to the maximum value of the response times with respect to all critical-instant candidates as shown in Eq. (8.20).

$$RT^l_i = \max_{\forall m_c \& \forall k} \{RT^l_{ic[k]}\} \tag{8.20}$$

8.5.4 Calculations for Queuing Jitter of Classes A and B

The arrival of a message from class A or B on a link that is shared with the class BE message (m_i) can vary due to traversal of the class A and B messages through multiple links in the network. This variation results in the queuing jitter of the A and B messages. This jitter may have a significant effect on the response time of m_i as shown in the calculations for higher-priority interference in Eq. (8.14) and Eq. (8.19). To calculate the jitter on link l, we should find the worst-case and best-case delays from the sender end station to the link l where it is shared by m_i. The worst-case RTA of classes A and B messages are already presented in Section 8.4. The best-case response time of classes A and B can occur by assuming that the corresponding credits are always available for the messages upon their arrival for transmission. Thus their best-case response times will be equal to their corresponding transmission times. Eq. (8.21) shows the calculations for release jitter of class A and B messages (m_j) on link l.

$$\begin{aligned} J_j^l &= \sum_{L=1..l} RT_j^L - \sum_{L=1..l} BT_j^L \\ \sum_{L=1..l} BT_j^L &= l.C_j \end{aligned} \tag{8.21}$$

8.5.5 Response Time over Multiple Links

As messages across multiple links are buffered in the queues of each switch, the worst-case response time of a message crossing multiple links is the sum of the per-link worst-case response times. Eq. (8.22) shows the worst-case response time of m_i in class BE crossing multiple links from its source till its destination end station. Note that the switch hardware latency γ is added for each link.

$$RT_i = \sum_{l=1..|\mathcal{L}_i|} (RT_i^l) + (|\mathcal{L}_i| - 1).\gamma \tag{8.22}$$

8.6 Evaluation

This section presents the evaluation of the presented response time analysis on a vehicular application use case. The main intention of the evaluation is to show that a network designer can evaluate the number of deadline misses in the BE traffic using the proposed analysis. Based on this information, the overall network configuration can be adjusted and refined.

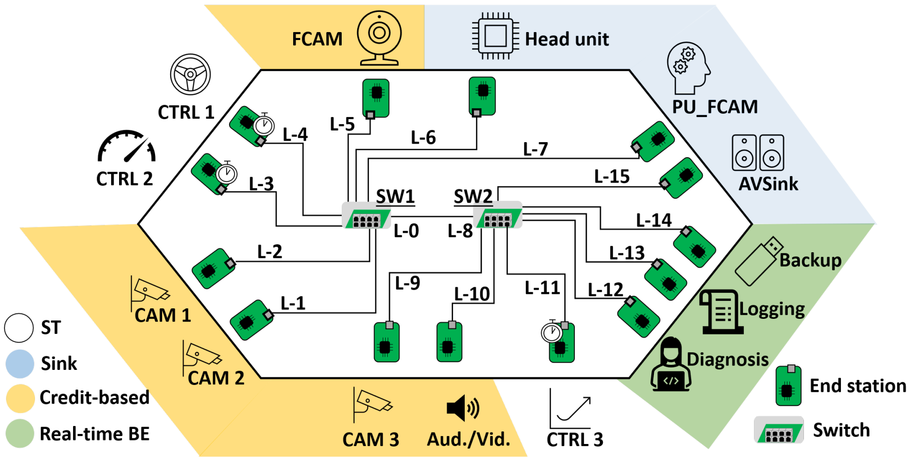


Figure 8.6. Vehicular application use case topology.

8.6.1 Vehicular Application Use Case

The use case is inspired from a most-commonly used industrial case study on a modern car developed in [23]. Based on the presented use case, we define a TSN network that consists of 14 end stations connected through two TSN switches, as shown in Figure 8.6. In this use case, we assume that the total network bandwidth is 10Mbps and the fabrication delay in all switches is assumed to be $5\mu s$.

In this use case, there are three cameras: CAM1, CAM2, and CAM3. These cameras are mounted on different sides of the car and send video streams using classes A and B. The receiver of the video streams is the Head Unit, which is the main (central) ECU in the use case. Moreover, FCAM is the main front camera of the car that sends the video streams to the PU_FCAM for further processing. Similarly, the Aud/Vid node transmits infotainment streams to the AVSink node. Both FCAM and Aud/Vid use classes A and B for the data transmission. In addition, we consider three control nodes (e.g., engine control) that send control signals to the Head Unit. The control end stations are denoted by CTRL1-CTRL3. The control signals are allocated to the ST class as we want to obtain full deterministic behaviour for them. Finally, we considered three end stations, being Diagnosis, Logging and Backup that transmit diagnostics information, stored data from other ECUs, and software update information. The data from these end stations are sent to the Head Unit. Table 8.1 shows the traffic characteristics.

As there are several class A and B messages, the values of the idleSlope for these

Table 8.1. Various traffic in the vehicular application use case.

ID	Sender	Receiver	$T = D$ (μs)	Payload ($Bytes$)	$C(\mu s)$	Class
M-0	Head unit	CTRL1	20000	20	49.6	ST
M-1	Head unit	CTRL2	20000	20	49.6	ST
M-2	Head unit	CTRL3	20000	20	49.6	ST
M-3	Head unit	CAM1	10000	786	662.4	A
M-4	Head unit	CAM2	10000	786	662.4	A
M-5	Head unit	CAM3	10000	786	662.4	A
M-6	PUFCAM	FCAM	5000	786	662.4	A
M-7	Head unit	CAM1	10000	786	662.4	B
M-8	Head unit	CAM2	10000	786	662.4	B
M-9	Head unit	CAM3	10000	786	662.4	B
M-10	AVSink	Aud./Vid.	5000	1472	1211.1	B
M-11	Head unit	Backup	10000	500	433.6	BE
M-12	Head unit	Logging	10000	1500	833.5	BE
M-13	Head unit	Diagnosis	10000	2000	1233.5	BE

Table 8.2. idleSlope of class A and class B per link.

idleSlope ($Mbps$)	l_1	l_2	l_5	l_6	l_7	l_8	l_9	l_{10}	l_{15}
Class A	1.32	1.32	2.64	3.97	2.64	1.32	1.32	0	0
Class B	1.32	1.32	3.97	3.97	0	1.32	1.32	4.84	4.84

two classes need to be allocated on every link through which these messages traverse. In order to calculate the idleSlope values, we use recommendation from the IEEE 802.1Q-2018 standard. Therefore, the idleSlope is equal to the utilization of the traffic with scaling up by considering the ST traffic. Table 8.2 shows the idleSlope values assigned to each link, where the link IDs are shown in Figure 8.6. Note that zero credit means there are no messages from the associated CBS class on the link.

8.6.2 Analysis Results

We implemented the proposed analysis as an in-house analysis engine. The configuration and message set in the vehicular application use case were fed as input to the implemented analysis. The analysis took 15 milliseconds to run on an HP Elitebook with an Intel Core i5 processor and a 16-Gigabyte RAM. Table 8.3 shows the worst-case response times of all messages calculated by the implemented analysis. As it can be seen, the messages from classes ST, A and B meet their deadlines. However, two of the BE messages (M-12 and M-13) miss their deadlines as their response times ($10674\mu s$ and $11074\mu s$) exceed their deadlines ($10000\mu s$). Note that the presented analysis calculates the response times of messages per link. Hence, the analysis can identify the most congested links that may become bottleneck in the TSN network. The network designer can use this information to check whether it is acceptable for the class BE messages to miss their deadlines with a small margin, otherwise a new configuration should be setup to refine the application by considering the bottlenecks indicated by the proposed analysis.

Table 8.3. Calculated response times (RT) in the use case.

ID	$T = D(\mu s)$	$RT(\mu s)$	Schedulable
M-0	20000	104.2	Yes
M-1	20000	104.2	Yes
M-2	20000	158.8	Yes
M-3	10000	3808.7	Yes
M-4	10000	3808.7	Yes
M-5	10000	5792.9	Yes
M-6	5000	1329.8	Yes
M-7	5000	2427.4	Yes
M-8	10000	5843.9	Yes
M-9	10000	5843.9	Yes
M-10	10000	8506.5	Yes
M-11	10000	9474.0	Yes
M-12	10000	10674.0	No
M-13	10000	11074.0	No

8.7 Conclusion and future work

In this paper, we argued that it is simpler to use BE class in TSN networks within a vehicle instead of classes A and B for the traffic with no hard real-time timing requirements. This is due to the complexity of the CBS configuration when it is combined with the TAS and frame preemption mechanisms in TSN. However, this requires us to provide a level of QoS for the BE traffic, thus a schedulability analysis for the BE class becomes essential. This paper presented the worst-case response-time analysis for the BE traffic in TSN while considering the effects of classes ST, A and B via the TAS and CBS, and the frame preemption support. To the best of our knowledge, this is the first schedulability analysis for the BE class in TSN networks. We used the presented analysis on a vehicular application use case to show how the analysis can provide essential information to the network designer. The future work entails using the analysis for BE traffic to improve the offline schedules for the ST traffic in TSN networks.

Acknowledgements

The work presented in this paper was supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the DESTINE, PROVIDENT and INTERCONNECT projects and the Swedish Knowledge Foundation via the DPAC and HERO projects.

Bibliography

[1] L. Lo Bello, R. Mariani, S. Mubeen, and S. Saponara. Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems. *IEEE Transactions on Industrial Informatics*, 2019.

[2] M. Ashjaei, L. Lo Bello, M. Daneshtalab, G. Patti, S. Saponara, and S. Mubeen. Time-Sensitive Networking in Automotive Embedded Systems: State-of-the-Art and Research Opportunities. *Journal of Systems Architecture*, 2021.

[3] L. Lo Bello, M. Ashjaei, G. Patti, and M. Behnam. Schedulability Analysis of Time-Sensitive Networks with Scheduled Traffic and Preemption Support. *Journal of Parallel and Distributed Computing*, 2020.

[4] M. A. Ojewale, P. M. Yomsi, and B. Nikolić. Multi-level Preemption in TSN: Feasibility and Requirements Analysis. In *IEEE International Symposium on Real-Time Distributed Computing*, 2020.

[5] L. Zhao, P. Pop, Z. Zheng, H. Daigmorte, and M. Boyer. Latency Analysis of Multiple Classes of AVB Traffic in TSN with Standard Credit Behavior using Network Calculus. *IEEE Transactions on Industrial Electronics*, 2020.

[6] U. D. Bordoloi, A. Aminifar, P. Eles, and Z. Peng. Schedulability Analysis of Ethernet AVB Switches. In *International Conference on Embedded and Real-Time Computing Systems and Applications*, 2014.

[7] D. Thiele, R. Ernst, and J. Diemer. Formal Worst-Case Timing Analysis of Ethernet TSN's Time-Aware and Peristaltic shapers. In *IEEE Vehicular Networking Conference*, 2015.

[8] D. Thiele and R. Ernst. Formal Worst-Case Performance Analysis of Time-Sensitive Ethernet with Frame Preemption. In *IEEE International Conference on Emerging Technologies and Factory Automation*, 2016.

[9] M. Ashjaei, G. Patti, M. Behnam, T. Nolte, G. Alderisi, and L. Lo Bello. Schedulability Analysis of Ethernet Audio Video Bridging Networks with Scheduled Traffic Support. *Real-Time Systems*, 2017.

[10] M. A. Ojewale, P. M. Yomsi, and B. Nikolić. Worst-Case Traversal Time Analysis of TSN with Multi-level Preemption. *Journal of Systems Architecture*, 2021.

[11] J. A. R. De Azua and M. Boyer. Complete Modelling of AVB in Network Calculus Framework. In *Proceedings of the International Conference on Real-Time Networks and Systems*, 2014.

[12] M. Boyer, H. Daigmorte, N. Navet, and J. Migge. Performance Impact of the Interactions between Time-triggered and Rate-constrained Transmissions in TTEthernet. In *European Congress on Embedded Real Time Software and Systems* , 2016.

[13] L. Zhao, P. Pop, Z. Zheng, and Q. Li. Timing Analysis of AVB Traffic in TSN Networks using Network Calculus. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, 2018.

[14] L. Zhao, P. Pop, and S. S. Craciunas. Worst-Case Latency Analysis for IEEE 802.1 Qbv Time Sensitive Networks using Network Calculus. *IEEE Access*, 2018.

[15] L. Maile, K. Hielscher, and R. German. Network Calculus Results for TSN: An Introduction. In *Information Communication Technologies Conference*, 2020.

[16] T. L. Mai, N. Navet, J. Migge. A Hybrid Machine Learning and Schedulability Analysis Method for the Verification of TSN Networks. In *International Workshop on Factory Communication Systems*, 2019.

[17] T. L. Mai, N. Navet, J. Migge. On the Use of Supervised Machine Learning for Assessing Schedulability: Application to Ethernet TSN. In *Proceedings of the International Conference on Real-Time Networks and Systems*, 2019.

[18] T. L. Mai and N. Navet. Improvements to Deep-learning-based Feasibility Prediction of Switched Ethernet Network Configurations. In *International Conference on Real-Time Networks and Systems*, 2021.

[19] J. Cao, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien. Tight Worst-case Response-time Analysis for Ethernet AVB using Eligible Intervals. In *IEEE World Conference on Factory Communication Systems*, 2016.

[20] D. Maxim and Y-Q Song. Delay Analysis of AVB Traffic in Time-Sensitive Networks (TSN). In *Proceedings of the International Conference on Real-Time Networks and Systems*, 2017.

[21] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen. Synthesising Schedules to Improve QoS of Best-effort Traffic in TSN Networks. In *International Conference on Real-Time Networks and Systems*, 2021.

[22] J. Maki-Turja and M. Nolin. Fast and Tight Response-Times for Tasks with Offsets. In *Euromicro Conference on Real-Time Systems*, 2005.

[23] H. Lim, K. Weckemann, and D. Herrscher. Performance Study of an In-car Switched Ethernet Network without Prioritization. In *Communication Technologies for Vehicles*, 2011.

Chapter 9

Paper D: Supporting End-to-end Data Propagation Delay Analysis for TSN-based Distributed Vehicular Embedded Systems

Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Saad Mubeen.
In the Journal of Systems Architecture (JSA), 2023.

Abstract

In this paper, we identify that the existing end-to-end data propagation delay analysis for distributed embedded systems can calculate pessimistic (over-estimated) analysis results when the nodes are synchronized. This is particularly the case of the Scheduled Traffic (ST) class in Time-sensitive Networking (TSN), which is scheduled offline according to the IEEE 802.1Qbv standard and the nodes are synchronized according to the IEEE 802.1AS standard. We present a comprehensive system model for distributed embedded systems that incorporates all of the above mentioned aspect as well as all traffic classes in TSN. We extend the analysis to support both synchronization and non-synchronization among the ECUs as well as offline schedules on the networks. The extended analysis can now be used to analyze all traffic classes in TSN when the nodes are synchronized without introducing any pessimism in the analysis results. We evaluate the proposed model and the extended analysis on a vehicular industrial use case.

9.1 Introduction

Vehicular distributed embedded systems are often modeled with chains of tasks and messages that can be distributed over two or more Electronic Control Unit (ECUs)[1] connected by a real-time network [1]. Traditionally, the in-vehicle communication was based on low-bandwidth and low-latency networks like Controller Area Network (CAN) [2]. Since CAN is an event-triggered communication protocol, the ECUs connected to the network are not synchronized. The traditional in-vehicle networks are unable to support high-bandwidth requirements in many complex vehicular distributed systems, in which realization of a higher level of autonomy of driving is envisioned. Realizing these vehicular systems is conditioned to incorporating a spectrum of functionalities, that range from handling high data-rate sensor readings, to gathering information of the vehicle's environment, and providing predictable responses to the corresponding inputs [3]. The communication standards utilized in future vehicular systems need to be flexible to allow accommodation of the new functions to the system over time. Recently, the IEEE Time-sensitive Networking (TSN) task group[2] developed a set of TSN standards that have emerged as a promising solution to support high-bandwidth and low-latency in-vehicle communication [4].

In vehicular distributed embedded systems, the data in a chain of tasks and messages propagates from the input to the output of the chain. The input corresponds to the first task in the chain, e.g., the task reading a sensor signal. Whereas, the output corresponds to the last task in the chain, e.g., the task producing an actuation signal. Note that any two neighbouring tasks within the chain communicate using over-writable and non-consuming buffers, also called registers. This means, the writer task can over-write the previous data in the buffer, whereas the data stays in the buffer after it is read by the reader task. If tasks in such a chain are activated by independent activation sources with different periodicity (e.g., different periodic clocks), the data can propagate through more than one path from the input to the output of the chain. This leads to different types of delays that the data can experience while traversing through the chain. These delays are called data-propagation delays or end-to-end delays. The developers of the systems are required to verify, at the design time, that the specified timing constraints are satisfied. This can be achieved by performing the end-to-end data-propagation delay analysis of these systems [5, 6, 7, 8].

A joint effort from the automotive[3] industry and academia identified the significance of these delays in vehicular systems and provided their formal semantics [5, 6, 9].

[1]We use the terms ECU, end-station and node interchangeably to refer to a single-core compute unit.

[2]https://1.ieee802.org/tsn

[3]Automotive systems are a subset of vehicular systems that include cars, trucks, construction vehicles, loading vehicles, moving cranes, to mention a few.

Eventually, the timing constraints corresponding to these delays were included in the automotive domain-specific modelling language EAST-ADL [10] and the AUTomotive Open System ARchitecture (AUTOSAR) standard [9]. In order to verify these timing constraints, the research community in collaboration with the R&D of the automotive industry developed the end-to-end data-propagation delay analysis [5, 6, 7, 8]. This analysis is already implemented in several tools that are used in the vehicle industry, e.g., SymTA/S[4][11] and Rubus [12]. Since CAN is the most widely used onboard real-time network in the vehicular domain, the existing analysis incorporated the response-time analysis for CAN [13, 14] within the end-to-end data-propagation delay analysis. Furthermore, as CAN is an event-triggered network communication protocol and it does not support synchronization of the connected ECUs, the data-path computation algorithm within the existing end-to-end data-propagation delay analysis did not account for synchronization of ECUs.

The existing end-to-end data-propagation delay analysis and its data-path computation algorithm also support the precursor of TSN, called the Ethernet Audio-Video Bridging (AVB), which includes some classes of TSN. This is because AVB also supports event-triggered traffic and does not consider synchronization of ECUs. In that case, the response-time analysis of AVB [15] was incorporated within the end-to-end data-propagation delay analysis [16]. However, the TSN standards support synchronization of ECUs according to the IEEE 802.1AS standard. In particular, the ECUs should be synchronized when the Scheduled Traffic (ST) class in TSN is used, according to the IEEE 802.1Qbv standard. The real-time traffic mapped to the ST class is transmitted according to a schedule that is created offline. In general, when the IEEE 802.1AS standard is used then the ECUs in the TSN network should be considered synchronized regardless of which TSN traffic class, ST, AVB and Best-Effort (BE), is used. We identify that when the data-path calculation algorithm in the existing end-to-end data-propagation delay analysis, i.e. [5], is applied to the case of synchronized ECUs, the analysis results can be pessimistic (over-estimated) because the algorithm does not consider synchronization among the ECUs.

9.1.1 Paper Contributions

In this paper, we extend the data-path computation algorithm within the existing end-to-end data-propagation delay analysis to support both synchronization and non-synchronization among the ECUs. The extended algorithm supports all traffic classes in TSN. Using the extended algorithm, the existing end-to-end data-propagation delay analysis can now be used to analyze all traffic classes in TSN networks, where the ECUs

[4]SymTA/S tool has been acquired by Luxoft (`https://www.luxoft.com`)

may or may not be synchronized, without introducing any pessimism (over-estimation) in the the analysis results.

The **main contributions** in the paper are as follows:

- We extend the data-path computation algorithm within the existing end-to-end data-propagation delay analysis to support all traffic classes in TSN networks when the ECUs are synchronized using the IEEE 802.1AS standard. Unlike the existing algorithm, the analysis results with the extended algorithm do not include any pessimism when the ECUs in the TSN networks are synchronized. The extended algorithm is backwards compatible to support the analysis of all non-scheduled traffic (non-ST) classes[5] (AVB or BE) when the ECUs are not synchronized in the TSN networks.
- We present a comprehensive system model for distributed embedded systems to support the extended algorithm, which incorporates all traffic classes in TSN. The model can express distributed task chains that can contain various types of traffic supported by TSN, including the ST, AVB, and BE traffic.
- We demonstrate the applicability of the presented model and analysis to a vehicular industrial use case. We also perform comparative evaluation of the extended analysis with the existing analysis by analyzing the use case with the two analyses. Furthermore, the presented model and analysis are evaluated by experiments to show the effect of various configurations of ST class, receiver periods, and synchronization of the sender and receiver ECUs on the end-to-end data propagation delays.

9.1.2 Paper Layout

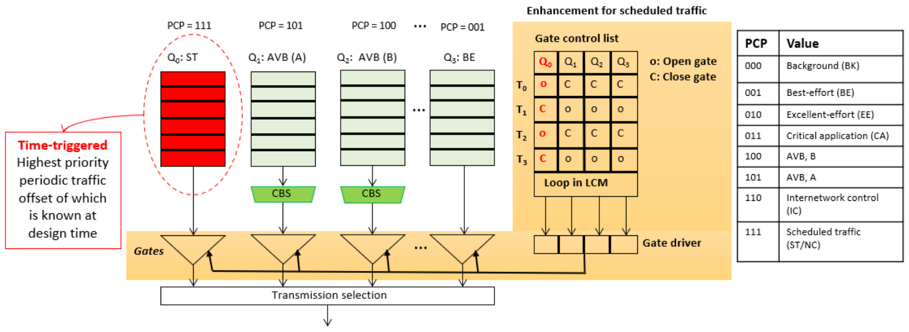
The rest of the paper is organized as follows. In Section 9.2, we provide background on TSN and related works on timing analysis of TSN networks. Section 9.3 describes the end-to-end data propagation delays and elaborates on the over-estimation of delays if the existing analysis is applied to the ST class in TSN. Section 9.4 presents the proposed system model for distributed embedded systems, and furthermore Section 10.3 presents the extension to the existing end-to-end data propagation delay analysis. Section 9.6 presents a vehicular application case study. We compare the results of the existing analysis with our proposed extended end-to-end data propagation delay analysis. Furthermore, we present the use-case results and an experimental study to show the effect of various parameters on the end-to-end data propagation delays. Finally, we discuss the results in Section 9.6.4, and in Section 9.7 we conclude the paper.

[5]It is required to use synchronization when the ST traffic class in TSN is used.

9.2 Background and Related Work

9.2.1 Time-Sensitive Networking (TSN)

TSN standards are recently developed by the TSN task group in IEEE standardization. This set of standards can be seen as a toolbox containing various features to improve the performance of communication in several applications, e.g., automation and automotive applications [1, 17, 18]. According to the IEEE 802.1Q-2022 standards, the traffic classes are categorized into three categories of ST, AVB, and BE traffic. Among several features, the TSN standards allow temporal isolation of the ST traffic that is transmitted according to an offline schedule via the Gate Control List (GCL) as shown in Figure 9.1. GCL is part of the Time-aware Shaper (TAS) that can realize the temporal isolation using a set of gates that control the transmission of traffic on a port of a TSN interface or switch. The gates can stop the transmission of lower priority traffic in favor of the urgent ST traffic class which in turn guarantees low-jitter transmission for the ST traffic (also known as preemption). In addition, the TSN standards define a Credit-Based Shaper (CBS) mechanism that allows reservation of bandwidth over the network for a set of traffic classes, known as the AVB classes. AVB traffic includes several classes starting from A (high priority) and can be up to eight as the number of queues per port is eight. It is very common to use only classes A and B in analysis and examples, while it is possible to have eight AVB traffic classes in the standard. Queues assigned to AVB class undergo the CBS mechanism for transmission. According to the CBS mechanism, a credit is configured per class of traffic on each TSN port and the traffic associated to the class can only be transmitted when the credit for that class is zero or positive. If the credit is negative, the transmission is on hold until the credit replenishes with a constant rate, known as the *idleSlope*, to zero or positive. The credit decreases when the transmission is happening with a constant rate, known as the *sendSlope*, and the summation of both values is equal to the port rate. Moreover, TSN can support legacy traffic transmission that do not need any timing guarantees, which is known as the BE traffic class. As shown in the example in Figure 9.1, the eight traffic priorities in the Priority Code Point (PCP) table are mapped to the eight configurable queues at the egress port of the TSN switch. A queue can be configured to use each of the aforementioned mechanisms for passing the traffic with the associated PCP to the port. In this example, the PCP ”111” is configured to use the ST class, PCP ”101” and PCP ”100” are assigned to AVB class with the priorities A and B subsequently. The PCP ”001” is considered as BE traffic.



PCP	Value
000	Background (BK)
001	Best-effort (BE)
010	Excellent-effort (EE)
011	Critical application (CA)
100	AVB, B
101	AVB, A
110	Internetwork control (IC)
111	Scheduled traffic (ST/NC)

Figure 9.1. TSN interface.

9.2.2 Related Work

Several schedulability analysis techniques have been proposed in the literature to calculate the worst-case delays of traffic crossing through a TSN network. Among the techniques, many of them focused on the worst-case delays of the classes A and B frames under the CBS only, e.g., the work in [19] and the improved technique given in [20]. In addition, the work in [21] proposed a technique based on the trajectory approach to compute the delays of classes A and B. The technique obtains tighter bound of delays compared to the previous techniques, e.g., compared to the delays calculated by the approach in [19]. Later, the work in [22] proposed the notion of eligible interval that could provide a bound for delays per frame that leads to tighter analysis compared to the previous analysis techniques. The above-mentioned works solely consider the CBS in TSN networks. However, the TSN standards give various number of shapers and mechanisms that a network designer can select from. For instance, the proposal in [23] and [24] presented an analysis based on network calculus where it considers a TSN shaper called the Burst-Limiting Shaper (BLS).

Various schedulability analysis techniques focused on mechanisms other than the CBS, such as the gate mechanism. The analysis that is proposed in [25] computes the worst-case response times of classes A and B messages in TSN considering both CBS and the gate mechanism. However, the analysis considers a single-switch network, while industrial networks can consist of multiple switches. The work in [26] presented an analysis for calculating the accumulation of delays per link, whereas the works in [27] and [28] used network calculus to check the schedulability of TSN messages. In addition, the technique in [29] presented a response-time analysis for classes A and B messages considering the CBS and support for the ST that was earlier proposed in [30].

Furthermore, the work in [31] extends the technique in [29] for supporting the BE traffic in the response-time analysis. The traffic forwarding and shaping model in the latter work was different than the TSN standard models, as it was proposed before finalization of the first TSN draft.

A TSN network may also benefit from the preemption support along with the CBS and gate mechanisms according to the IEEE Time-sensitive Networking (TSN) task group[6]. Therefore, the work in [32] proposed an analysis considering frame preemption under the IEEE 802.3br standard. Further, the work in [33] presented a technique that calculates the worst-case response times of frames for classes A and B when the CBS, gate mechanism and frame preemption are used in a TSN network. Similarly, the work in [34] proposed a response-time analysis with the mentioned TSN features in combination with various modes, such as enabling and disabling the hold and release mechanism. The hold and release mechanism is defined in the TSN standards to prevent any possible jitter for the ST traffic due to transmission of lower-priority classes A and B.

To verify the timing behavior of TSN-based distributed embedded systems, not only the response times of tasks in the nodes and messages in the TSN network should be taken into account, but also the end-to-end data propagation delays in the chains of tasks that include TSN messages should be considered.

According to the classification in reference [35], there are three approaches for calculating the worst-case end-to-end delays: 1) simulation-based method that obtains the maximum end-to-end delays from a set of selected scenarios. The simulation approach does not necessarily show the maximum end-to-end delay, because it might not cover all the possible assumptions for the worst-case scenario; 2) model-checking based on an exhaustive search can provide exact worst-case end-to-end delays even on large-scale networks. However, these methods have high time complexity; and 3) analytical approach that provides upper bounds on end-to-end delays with a certain pessimism. Our work lies in the group of analytical approaches in the above mentioned classification.

From a different perspective, the reference [36] mentions active and passive approaches for the end-to-end data propagation delay analysis. Active approaches optimize the pattern of task releases in a chain to achieve optimal delays. Whereas, passive approaches study the worst-case assumptions for the end-to-end delays in distributed embedded systems to find upper bounds on the delays. This paper and the works in [5, 36, 37, 38, 7, 39, 40, 8] are among works in the passive category. In the following paragraphs of this section, we present some of the recent works on the end-to-end data propagation delay analysis.

[6]https://1.ieee802.org/tsn

The existing end-to-end data propagation delay analysis that computes the end-to-end delays incorporates the response-time analysis of various legacy real-time networks, such as CAN [7] and legacy Ethernet [41].

The work in [42] targets data age delay in cause-effect chains within one execution node. The tasks are synchronized inside an end-station, and they are scheduled with offsets. The aim of the paper is to find priorities, offsets and to optimize the design mapping for tasks to minimize the data age delays in a chain of tasks. Similarly, the work in [43] aims at finding offsets for the chain of tasks to optimize the end-to-end delays. The works in [42, 43] belong to the active end-to-end data propagation analysis. Moreover, the paper [44] studies the dependencies between the task instances. Such dependencies can be specified at early development stages to guarantee data age delay constraint.

The work in [36] performs end-to-end timing analysis for the systems with locally synchronized periodic tasks in one end-stations. The end-stations only support non-synchronized communication, i.e., via CAN, or FlexRay. Similarly, the work in [45] considers globally non-synchronized communication among the end-stations, while the tasks within the end-stations are considered synchronized. The authors in [45] propose a computationally-efficient analysis compared to the analysis in [36]. Besides, the work in [45] achieves a higher upper-bound on the data age delay than the work in [36].

The focus of the aforementioned works are only on data age delay, whereas our analysis also includes the analysis of the reaction delay. The reference [5] is a seminal work in the literature that introduces a formal framework for defining end-to-end delays in the periodic and register-based systems. The task model in [5] is based on Bounded Execution Time (BET) task model. In BET task model, the communication between the tasks is implicit where a task reads data from the register at its beginning and writes to the register at its end of execution. Furthermore, the work in the reference [46] proposes an end-to-end data propagation delay analysis that includes sporadic tasks based on BET task model. The work in paper [37] builds on the Logical Execution Time (LET) paradigm by proposing a system-level LET-based communication model for distributed embedded systems. LET is an inter-task communication model that augments the read/write access times (input and output of the task) to the task's physical execution time model. System-level LET models the communication between tasks of different end-stations. In the work presented in [37], the end-stations have their own local clocks (timeline). The global timeline is approximated based on the local timeline of the sender and receiver end-stations with a bounded error. In our end-to-end data propagation delay analysis, we rely on the determinism promised for ST traffic, and exclude the global synchronization error for analyzing transactions that utilize ST traffic. The work in [47] considers globally non-synchronized and locally synchronized task chains and propose a method to calculate a limited number of timed-paths[7] that lead to the maximum end-to-end delays. We extend the timed-path approach based on the works in [8, 41, 7, 5], which calculate all possible timed-paths but within a bounded window equal to twice the hyperperiod or the Least Common Multiple (LCM) of the periods of all involved tasks in the chain. Consequently, our algorithm finds the timed-path that leads to the worst-case end-to-end delays after considering all possible cases.

While the majority of the works focus on providing methods for calculating the maximum end-to-end delays in the chains, the work in [48] discusses robustness margins around the end-to-end timing constraints. The work in [48] employs BET and system-level LET communication model, and further studies the variations in the tasks' response times and the influence on the robustness of the system (i.e, variations in the end-to-end delays). In our evaluations, we also take into consideration the changes in the end-to-end timing delays by making variations in the configuration of ST traffic and the periods of the receiver tasks. We focus only on the BET and consider pre-defined end-to-end timing constraints for the system under evaluation.

The work in [38] proposes an end-to-end data propagation delay analysis for the chains of tasks in the context of Robot Operating Systems (ROS). The analysis prior to [38] mainly focused on analyzing periodic and sporadic tasks. The work in [38] extends the end-to-end timing analysis to support ROS2 task chains that deal with a mix of time-triggered and event-triggered functions. Besides, the network model in [38] is based on publisher-subscriber communication model. The communication model is therefor inherently non-synchronized. The proposed analysis in our work is only applicable to periodic tasks.

In comparison to the aforementioned works, this paper aims at extending the data-path calculation algorithm within the existing end-to-end data propagation delay analysis [5, 7, 8, 41, 47, 49] to support analysis of all traffic classes in TSN where nodes can be synchronized or non-synchronized. These works have also been implemented in tools to support model- and component-based software development of vehicular embedded systems, e.g., [7, 39, 40], all of which are considering the BET task modeling paradigm.

The existing timed-path calculation used in the end-to-end data propagation delay analysis algorithms, such as [5, 47], applies to traffic classes that do not require offline schedules, i.e., AVB and BE. For example, the work in [47] provides end-to-end analysis for the synchronized communication between sender and receiver end-stations, but it only considers the case when non-ST are transmitted between synchronized end-stations, i.e. AVB or BE.

[7]We use the terms timed-paths and data-paths interchangeably to refer to the path that the data traverses from one task to another within the task chain.

To the extent of our knowledge, there are a few works that aim at realizing end-to-end data propagation delay analysis for interconnected end-stations in distributed embedded systems that are based on the system-level LET model, such as [37, 36, 38, 48]. Unlike the previous works, this paper considers that the ST messages are scheduled with offsets (with globally synchronized end-stations).

These two models can be mapped to each other according to [48]. We chose to use the BET model because it is already integrated to several tools (including industrial tools) that support model- and component-based software development of vehicular embedded systems, e.g., in [39, 40].

9.3 End-to-end Data Propagation Delays

Embedded real-time systems are often modeled with chains of tasks and messages. To verify the timing behavior of these chains, not only their end-to-end response times need to be calculated and compared against the corresponding deadlines, but also the end-to-end data propagation delays (data age and reaction time) should be calculated and compared with the corresponding data age and reaction time constraints. The timing constraints on the data age and reaction delays are often specified on these distributed chains. The constraint on the data age delay is important, in particular, for control applications where freshness of the data is of value. Whereas, the reaction constraint is important in applications where the time of the first reaction to the input event is of value. These constraints are included in the timing model of the AUTOSAR standard [9] and are translated to several modeling languages in the vehicular domain [50].

9.3.1 Data Propagation Delays in Single-Node Embedded Systems

In order to explain the data age and reaction time delays, consider a task chain consisting of three tasks τ_1, τ_2 and τ_3, as shown in Figure 9.2. All tasks belong to a single-core node and are activated independently. The periods of activation for tasks τ_1, τ_2 and τ_3 are 8 ms, 8 ms and 4 ms, respectively. The Worst-Case Execution-Time (WCET) of each task is assumed to be 1 ms. For simplicity, we assume that the priority of τ_1 is higher than the priority of τ_2 and the priority of τ_2 is higher than the priority of τ_3. By this priority assignment policy, we ensure that the precedent elements in the chain should be executed before their subsequent elements in the chain. The tasks use register-based communication, i.e., they communicate with each other and with their environment by means of writing data to/ and reading data from the registers. The registers are of non-consuming type. This means that data stays in the register after the reader has read the data. Furthermore, the registers are over-writable, i.e., if the writer is faster than the reader then the previous data in the register can be overwritten by the new data before the reader can read the previous data. The data read by τ_1 from Reg-1 corresponds to the input of the chain. Similarly, the data written to Reg-4 by τ_3 corresponds to the output of the chain.

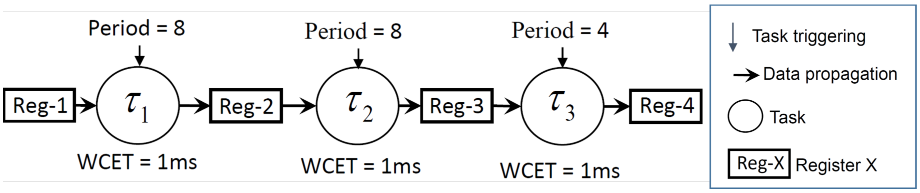


Figure 9.2. An example of a task chain that uses register-based communication.

As the tasks are activated independently and some tasks have different periods, the data traverses through the chain via multiple paths from the input to the output of the chain as shown in Figure 9.3. These paths are called timed-paths (also referred to as data-paths). Due to multiple timed-paths, there can be various delays to deliver the data from the input to the output of the chain.

The data age delay is the time elapsed between the arrival of data at the input and the latest availability of the corresponding data at the output. In the data age delay analysis, we are interested in identifying the longest time difference between the input data and the last sample of corresponding output data. On the other hand, the reaction delay corresponds to the earliest availability of the data at the first instance of the output corresponding to the data that *just missed* the read access at the input. An event (corresponding to availability of data) is considered as readable by an instance of a task, if it occurs at or before the activation of the task. If the event happens just after the activation of the task instance, the data is not readable to this instance, i.e., the data is just missed by the current instance of the task. The missed data is read by the next instance of the task. This is illustrated by the white thunderbolt in Figure 9.3, where the first instance of τ_1 at time 0 misses the data but the same data is read by the next instance of τ_1 at time 8.

Possible data age and reaction delays in the chain in Figure 9.2 are shown in Figure 9.3. On the one hand, the data from the event happening a bit before time 16 is accessible to the third instance of τ_1 (activated at time 16). In such a case, the latest impact of this event is available at the output of the chain until 5 ms after the occurrence of the event (data age delay). On the other hand, the sampling of the data coming from the event happening a bit after the time 0 is delayed until the time 8, where the data can be read by the second instance of τ_1. Accordingly, the earliest time the impact of the data appears at the output of the chain is 11 ms after the occurrence of the event (reaction time delay).

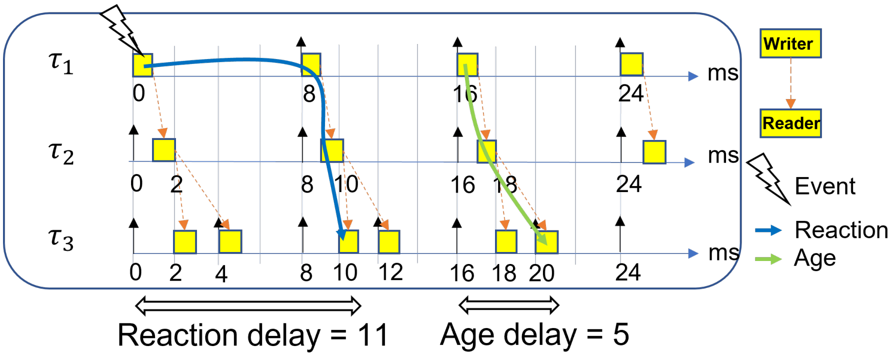


Figure 9.3. Data age and reaction time delays in the task chain depicted in Figure 9.2.

9.3.2 Data propagation Delays in Distributed Embedded Systems

The data propagation delays are equally valid in distributed embedded systems. Let us consider a distributed task chain in a distributed embedded system depicted in Figure 9.4, where two nodes are connected via a network. In this example, the tasks are activated periodically with periods of 6 ms and 3 ms, respectively. Task τ_1 in Node 1 sends a message to task τ_2 in Node 2 through the network.

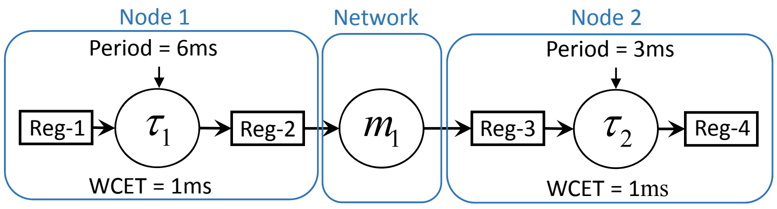


Figure 9.4. A multi-rate chain in a distributed embedded system.

Depending on the type of network, we may have different possible timed-paths through which the data can propagate from the sender task to the receiver task. For example, when the network is not capable of initiating communication independent of the sending tasks, a message can only be queued for transmission at the network interface by the sending task. This is the case of many event-triggered network protocols, like CAN [2]. In this case, the message inherits its period from the sender task. Furthermore, the timed-paths in a distributed task chain also depends upon whether the network supports synchronization of nodes. For example, TSN supports synchronization among the end stations via the IEEE 802.1AS standard, whereas the CAN protocol does not support synchronization. Figure 9.5 shows an execution trace when the nodes are synchronized in the system that is shown in Figure 9.4. The data age and reaction time delays in this distributed chain are identified as 7 ms and 10 ms, respectively.

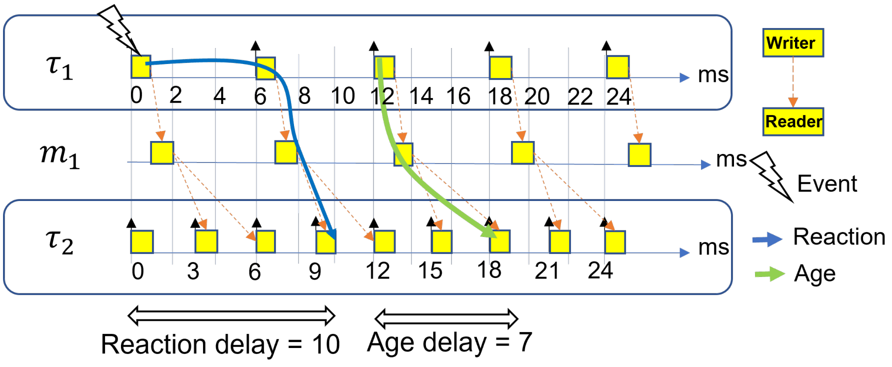
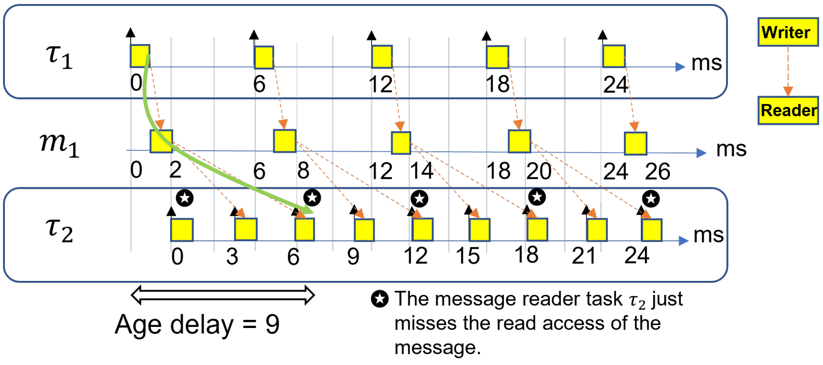


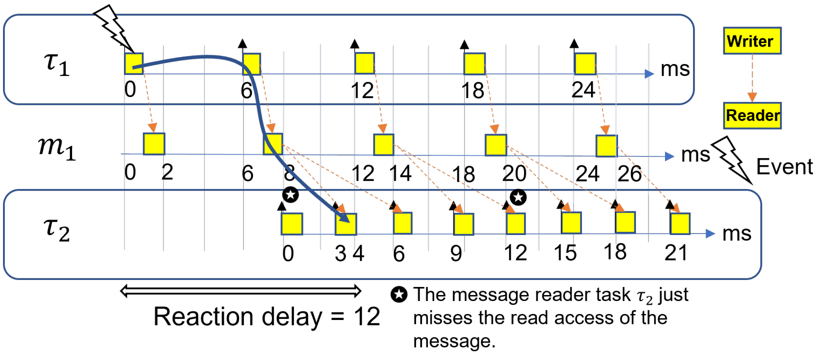
Figure 9.5. A possible execution trace for the distributed embedded system example shown in Figure 9.4 when source and destination end-stations are synchronized.

A possible execution trace of the distributed task chain in Figure 9.4 when the nodes are not synchronized is shown in Figure 9.6(a). To create worst-case conditions when the nodes are not synchronized, we assume that the receiver task τ_2 is activated "just before" the arrival of the message at the receiver node. Hence, the current instance of τ_2 (the first period activation) will miss the read access of the message. The message will be read by the next instance of τ_2 (second period activation) as shown in Figure 9.6(a). The corresponding data age delay is identified as 9 ms as shown in Figure 9.6(a).

To increase readability, we draw the same execution trace separately for the case of reaction time delay in the distributed task chain (as shown in Figure 9.4) when the nodes are not synchronized as depicted in Figure 9.6(b). In Figure 9.6(b), the first instance of τ_1 is activating the first instance of the message m_1. According to the assumption for the reaction delay, the first instance of τ_1 has missed the sampling of the chain's input event, thus the first instance of m_1 does not deliver valid data from the input event to the receiver task. However, as the second instance of τ_1 reads the input event, the second instance of the message m_1 also holds fresh data. Since τ_2 is not synchronized with τ_1, the worst-case assumption is that τ_2 is activated a small amount of time earlier than arrival of the message that holds the sampled data. Consequently, the first instance of τ_2 misses to read data of the event, which is being written by the second instance of m_1. But, at the next instance of τ_2 (second period activation), τ_2 is able to read the incoming data from m_1. Accordingly, the reaction delay is 12 ms.


τ_1
m_1
τ_2
ms
Writer
Reader
Age delay = 9
The message reader task τ_2 just misses the read access of the message.

(a) Data age delay.


τ_1
m_1
τ_2
ms
Writer
Reader
Event
Reaction delay = 12
The message reader task τ_2 just misses the read access of the message.

(b) Reaction time delay.

Figure 9.6. A possible execution trace for the distributed embedded system example shown in Figure 9.4 when source and destination end-stations are not synchronized.

9.3.3 Need for Extending the Timed-path Calculation Algorithm

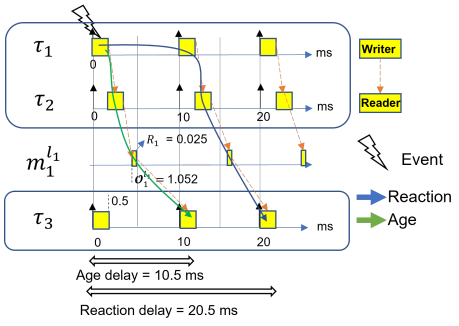
The timed-path computation algorithm in the existing end-to-end data propagation delay analysis implicitly assumes that the nodes are not synchronized. Hence, the worst-case assumption for the message receiving task is that the task is released for execution ”just before” the arrival of the message. This implies that the current instance of the task cannot read the message and hence the next instance of the task will read the message. This may not be true in the case of TSN when the nodes are synchronized using the IEEE 802.1AS standard. If the existing algorithm is applied to synchronized nodes, as is the case of TSN, then the calculated delays can be pessimistic (i.e., over-estimated). This can be seen in the figures Figure 9.5, Figure 9.6(a), and Figure 9.6(b), where data age and reaction time delays of 7 ms and 10 ms are calculated for synchronized end-stations (Figure 9.5). In comparison to the case of non-synchronized end-stations respectively (Figure 9.6(a), and Figure 9.6(b)), data age and reaction delays of 9 ms and 12 ms are calculated respectively.

Another example can be seen in Figure 9.7, where the existing end-to-end data propagation delay analysis is performed on a transaction between two end-stations that use ST class for transmission of message m_1 via link l_1 [8]. In the sender end-station, the periods of each of tasks τ_1 and τ_2 are 10 ms. The message m_1 is sent by τ_2 in the sender end-station. The period of the receiver task (τ_3) is 10 ms, and it reads the data coming from the message m_1 from the link l_1. The offset of the message is set to 1.052 ms, and the message is expected to finish its transmission at 0.025 ms later than the offset of the message at l_1. The WCET of each task is 0.5 ms. In the ideal condition for ST traffic when the sender and receiver end-stations are synchronized as shown in Figure 9.7(a), the data age and reaction time delays are subsequently 10.5 ms, and 20.5 ms. As seen in Figure 9.7(b), the data age and reaction time delays for this transaction in the case of non-synchronized nodes are pessimistic (over-estimated), i.e., 11.577 ms and 21.577 ms respectively. This is not desirable, since ST traffic is most commonly applied in critical applications that require precise calculation of the delays. Therefore, the data-path computation algorithm in the existing analysis requires extension to support synchronized nodes without introducing any pessimism in the analysis results.

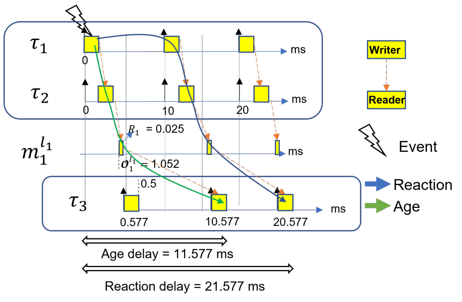
9.4 System Model

In this section, we formally present the system model of a distributed embedded system that consists of two or more end-stations (single-core nodes, compute units, or ECUs)

[8] A detailed system model with all notations will be described in Section 9.4.



(a) τ_1 and τ_2 are synchronized.



(b) τ_1 and τ_2 are not synchronized.

Figure 9.7. End-to-end delays for an ST message.

that are connected by a TSN network. The system $\mathcal{S}$ consists of a set of *transactions*, denoted by Γ, a set of *end-stations*, denoted by $\mathcal{E}$, and a *network*, denoted by $\mathcal{N}$. The system is formally expressed by the following tuple.

$$\mathcal{S} := \langle \Gamma, \mathcal{N}, \mathcal{E} \rangle \tag{9.1}$$

A transaction (denoted by Γ) represents the model of a distributed task chain that consists of two or more tasks. The chain of tasks either can be executed within one end-station; or tasks of different end-stations can communicate with each other via one or more messages in the network ($\mathcal{N}$). Multiple transactions can exist in the system model. The set of transactions are formally expressed subsequently by Eq. (9.2):

$$\Gamma := \{\Gamma_1, ..., \Gamma_{|\Gamma|}\} \tag{9.2}$$

The set of end-stations in the system are represented by Eq. (9.3).

$$\mathcal{E} := \{\mathcal{E}_1, ..., \mathcal{E}_{|\mathcal{E}|}\} \tag{9.3}$$

9.4.1 End-station Model

An end-station $\mathcal{E}_i$ may consist of one or more tasks as shown in Eq. (9.4):

$$\mathcal{E}_i := \{\tau_{ij1}, ..., \tau_{ijk}\} \tag{9.4}$$

where i is the index of the end-station to which the task belongs. Note that a task in an end-station may be a part of one or more transactions. Hence, j represents the transaction index. Finally, k represents the unique identifier of the task within the scope of the end-station.

9.4.2 Task Model

The properties of a task are specified by the tuple in Eq. (9.5).

$$\tau_{ijk} := \langle P_{ijk}, C_{ijk}, T_{ijk}, J_{ijk}, O_{ijk} \rangle \tag{9.5}$$

where, C_{ijk} is the task's WCET, T_{ijk} is the task's period, P_{ijk} is the task's priority, and J_{ijk} is the task's release jitter. Moreover, O_{ijk} represents the offset of the task.

Some other properties of the task are calculated using the aforementioned information in the task's tuple. Firstly, the activation time of the n^{th} instance of the task τ_{ijk} (denoted by $\alpha_{ijk}(n)$) can be obtained using the task's period and offset based on Eq. (9.6). Moreover the worst-case response time of the task is indicated by R_{ijk}. Also, the n^{th} instance of the task τ_{ijk} is denoted by $\tau_{ijk}(n)$.

$$\alpha_{ijk}(n) = n * T_{ijk} + O_{ijk} \tag{9.6}$$

9.4.3 Network Model

The network attributes are indicated by a set of parameters in Eq. (9.7):

$$\mathcal{N} := \langle s, \mathcal{L}, \mathcal{I} \rangle \tag{9.7}$$

where s is the overall network speed. We assume the network operates with the same speed on all of the links. $\mathcal{L}$ holds the set of links in the network. We consider that each link creates a bi-directional connection between an end-station and a switch or between two switches. All switches in the network are TSN switches, hence there can be different traffic classes in the network. We indicate the traffic classes by the set $\mathcal{I}$ in Eq. (9.8), where AVB can be classes A, B or other classes that undergo the CBS. Moreover, ST and BE represent the scheduled traffic and best-effort traffic classes respectively.

$$\mathcal{I} = \{AVB, ST, BE\} \tag{9.8}$$

A message in the network is indicated by m_{jk}, where the subscript j identifies the transaction to which the message belongs. Furthermore, the subscript k is the unique identifier of a message within the scope of the network. Eq. (9.9) shows the set of attributes defining the properties of a message.

$$m_{jk} := \langle P_{jk}, Size_{jk}, T_{jk}, J_{jk}, \mathcal{L}_{jk}, \mathcal{O}_{jk} \rangle \tag{9.9}$$

where the priority of the message is denoted by P_{jk} that specifies the TSN class from which the message is transmitted, e.g., class ST, AVB (A, B, or etc.) and BE. In this model, the ST class has the highest priority, while AVB has a lower priority than ST. Moreover, the BE class has the lowest priority among all the other classes. The size of data (payload size in bytes) is indicated by $Size_{jk}$. We assume that all messages are periodic, therefore T_{jk} is the period of the message. The release jitter of the message is indicated by J_{jk}. The set of links assigned as the route of the message from the source end-station to the destination end-station is stored in the set $\mathcal{L}_{jk}$. Furthermore, the set of offsets of the message at each of the links specified in the set $\mathcal{L}_{jk}$ is stored in the set $\mathcal{O}_{jk}$. We assume that we only know the offset of the ST traffic as it is scheduled offline. Therefore, the set $\mathcal{O}_{jk}$ for non-ST traffic (AVB or BE) is assumed to be empty, i.e., $\mathcal{O}_{jk} = \{\}$.

Based on the aforementioned properties, we can calculate other properties of the message m_{jk}, such as the transmission time (C_{jk}) as well as the worst-case response time (R_{jk}) by utilizing the response-time analysis of various classes in TSN [31]. Moreover, the activation time of the n^{th} instance of the message m_{jk} at link l is calculated by Eq. (9.10), where, O^l_{jk} is the offset of m_{jk} on link l.

$$\alpha^l_{jk}(n) = n * T_{ijk} + O^l_{jk} \tag{9.10}$$

We assume that both ST and non-ST traffic inherit the period from the corresponding sending task. Therefore, T_{ijk} indicates the period of the k^{th} task (sending task) belonging to the i^{th} end-station and being part of the j^{th} transaction. Moreover, we denote the n^{th} instance of the message m_{jk} by $m_{jk}(n)$.

9.4.4 Transaction Model

A transaction Γ_j represents the model of a distributed task chain that consists of two or more tasks that communicate with each other via one or more messages. The data read by the first task of the transaction is considered as the input of the transaction, and the data written by the last task of the chain corresponds to output of the transaction. The period of the transaction is denoted by T_j. Note that this model limits the number of message to one per transaction.

Figure 9.8 shows an example of a TSN-based distributed embedded system with the presented model. There are two end-stations that are connected to one TSN network. More specifically, end-station $\mathcal{E}_1$ and $\mathcal{E}_2$ are connected via links l_1 and l_2 to switch

1 ($SW1$). There are two transactions in the system, namely Γ_1 and Γ_2, as shown in Figure 9.8. These transactions are further elaborated in Figure 9.9.

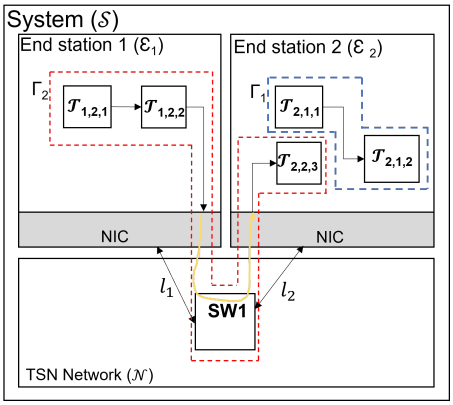


Figure 9.8. Example of a distributed vehicular embedded system based on TSN.

According to the example in Figure 9.8, Γ_1 is a transaction that is within a single end-station ($\mathcal{E}_2$) and only includes tasks from $\mathcal{E}_2$. Hence, $\mathcal{E}_2$ is both initiator and terminator end-station of Γ_1. As shown in Figure 9.9, transaction Γ_1 initiates and terminates inside the end-station $\mathcal{E}_1$. On the other hand, transaction Γ_2 is distributed over two end-stations. Γ_2 initiates from $\mathcal{E}_1$ and terminates in $\mathcal{E}_2$. The message $m_{2,1}$ travels from link l_1 and l_2 between the transaction initiator and terminator. It should be noted that, the transaction model does not consider forking and joining of tasks in a transaction.

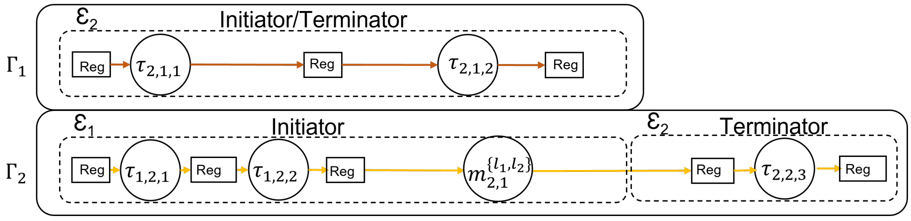


Figure 9.9. Example of transactions.

Trigger modes:

According to [51], the assumptions on the activation times of the tasks and messages in a distributed transaction affect the delays of the transaction. In this paper, we assume that each entity in a transaction, regardless of being a task or message, can be triggered in two modes, i.e. "$Dependent$" or "$Independent$". The trigger mode is selected by the parameter $triggerMode$, as shown in Eq. (9.11).

$$triggerMode := \{Dependent, Independent\} \tag{9.11}$$

A task can be independently triggered by an event source, e.g., a periodic clock. Additionally, a task can be triggered based on (i) an activation signal from a predecessor task, (ii) receiving data from a predecessor task, or (iii) a combination of both.

Message Activation Time:

If the message is ST, it is triggered independently. This means that the message is triggered based on static offsets defined for it at each of the links specified in its route to the destination end-station. In case of the dependent trigger mode, the message is triggered at the end of the execution of its sender task. For example, the message assigned to non-ST classes in TSN networks (AVB or BE) can use the link as soon as the message's sender task completes its execution as well as the bandwidth on the link is available, and there are no higher priority messages that need to be transmitted.

Receiver Task's Activation Time:

Activation time of the message receiving task can be calculated based on different assumptions. For instance, if the sender and receiver end-stations are synchronized, the first instances of their tasks are assumed to be activated simultaneously. In such a case, the receiver task polls for the TSN network to read data from the messages. Accordingly, the activation time of the receiving task within a synchronized network is calculated by the generic equation in Eq. (9.6). Where each instance n of the task within the receiving end-station is released periodically (T_{ijk}) and can have offset specified by the parameter O_{ijk}. In the non-synchronized network, the existing analysis assumes that the receiver task gets read access as soon as a message is available on the network. Hence, the activation time of the message receiving task is represented by Eq. (9.12).

$$\alpha_{ijk}(n) = n * T_{ijk} + O_{ijk} + R_{jk} \tag{9.12}$$

Transaction Constraints:

A timing constraint on transaction, denoted by Cr_j, defines the maximum allowed value of the delays, such as data age (Age_j) and reaction time $(Reac_j)$. Moreover, a deadline constraint (D_j) can be specified on a transaction. Deadline constraint corresponds to the end-to-end response time, i.e., response time of the transaction from its input to its output. These constraints are shown in Eq. (9.13) for transaction Γ_j:

$$Cr_j := \{Age_j, Reac_j, D_j\} \tag{9.13}$$

9.5 End-to-end data-propagation delay analysis

This section presents the end-to-end data propagation delays analysis of task chains that are distributed over TSN network. The analysis is based on the existing analysis [5, 7] that considers legacy networks like CAN. The end-to-end data propagation delay analysis requires computation of all relevant data-paths (also called reachable timed-paths) within the distributed task chains. In TSN unlike CAN, different traffic classes are to be analyzed in the same network, which would require incorporating synchronized and non-synchronized end-stations. The data-path computation algorithm in the existing end-to-end data propagation delay analysis [5, 7] only support non-synchronized end-stations. In this section, we propose an extended algorithm for TSN networks which covers all the TSN traffic characteristics while supporting both synchronized and non-synchronized end-stations.

9.5.1 Reachable Timed-paths

The order of read and write by each instance of the tasks from the input to the output of the transaction is represented by a set of timed-paths. These timed-paths track the propagation of data from the input to the output of the transaction. Therefore, each transaction can have a set of timed-paths. A timed-path belonging to the transaction Γ_j is denoted by tp_j^i, where i is the ID of the timed-path. A valid timed-path between a writer and reader task instance (the term ”instance” is equivalent to the term ”task job”) is selected according to a set of conditions as presented in [5]. In the following paragraphs of this section, we briefly explain the Boolean functions for checking the reachability of the timed-paths as presented in [5].

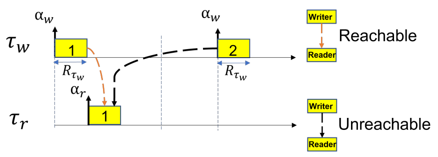
The first condition in identifying a valid timed-path checks whether the reader task (say τ_{dbe}) is activated after the activation of the current instance of the writer task (say τ_{abc}). Violating this condition is also known as activation time-travel $(att())$, therefore Eq. (9.14) defines this condition in negated form. Because, the activation time-travel should not happen in valid timed-paths. Note that $\alpha(w)$ shows the activation time of the w^{th} instance of the task computed according to Eq. (9.6).

$$att(\tau_{abc}(w) \longrightarrow \tau_{dbe}(r)) = \alpha_{dbe}(r) < \alpha_{abc}(w) \tag{9.14}$$

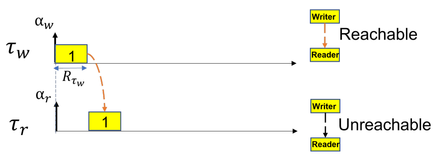
Eq. (9.14) is true in case the activation time-travel happens between the writer and reader tasks. Otherwise, for valid timed-paths it is desirable when the condition in Eq. (9.14) is false.

Moreover, the completion of the writer and reader tasks should not overlap in a valid timed-path. Therefore, the next reachability condition checks whether the activation time of the reader task is after the completion of the current instance of the writer task (according to R_{abc}, the worst-case response time of the writer task, referring to the system model presented in Section 9.4). Violating this condition is also known as critical condition and is represented by the critical function ($crit()$), as shown in Eq. (9.15). $crit()$ function returns true if the reader task τ_{abc} is activated before the writer task is completed. In such a case, the reader task misses the data from the writer task. Otherwise, the function in Eq. (9.15) returns false, which is desirable for valid timed-paths.

$$crit(\tau_{abc}(w) \longrightarrow \tau_{dbe}(r)) = \alpha_{dbe}(r) < \alpha_{abc}(w) + R_{abc} \tag{9.15}$$



(a) Different activation times.



(b) Activation at the same time.

Figure 9.10. Reachability conditions for tasks in the same end-station.

The instance number of each instance of writer and reader tasks are indicated in Figure 9.10(a) and Figure 9.10(b). Figure 9.10(a) shows an example of the case where the reader task is activated just after the the writer task is completed. In this case, we have a reachable timed-path from the first instance of the writer task to the first instance of the reader task. Note that the timed-path from the second instance of the writer task to the reader task's first instance is not reachable, and it will be excluded by Eq. (9.14) and Eq. (9.15).

Moreover, the reader and writer task instances can only overlap when they are activated at the same time in the same single-core end-station. In such a case, the reader task does not miss the writer task's data, if the priority of the reader task (P_{dbe}) is lower than the priority of the writer task (P_{abc}) as shown in Eq. (9.16). If the reader task executes before the writer task, then the reader task needs to wait until its next period activation in order to read the fresh data from the writer task. This is taken into account according to the wait function ($wait()$) in Eq. (9.16), where P indicates the priority of the task according to the system model in Section 9.4.1.

$$wait(\tau_{abc}(w) \longrightarrow \tau_{dbe}(r)) = P_{dbe} < P_{abc} \tag{9.16}$$

Figure 9.10(b) shows an example of a timed-path between two tasks inside an end-station. If both writer and the reader tasks are activated at the same time, there is a reachable timed-path between the writer and the reader task, provided that the writer task has executed before the reader task.

Accordingly, if the writer and reader task instances are from two different end-stations, the reachability of the timed-path through the network is obtained referring the worst-case response time of the message communicated between the writer and reader tasks, and the activation time of the writer task instance, as shown in Figure 9.11. For instance, if there is a writer task τ_w and a reader task τ_r which communicate by a message Msg, there are three different timed-paths as shown in Figure 9.11. However, only one of those timed-paths is a reachable timed-path from the input to the output of the transaction.

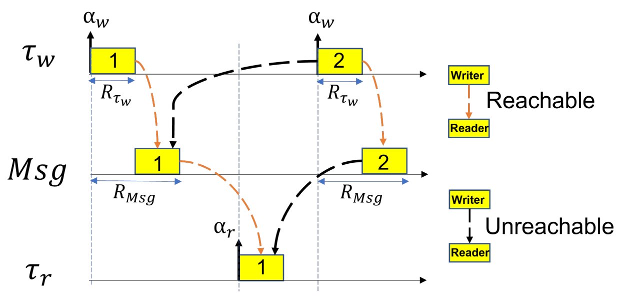


Figure 9.11. Timed-paths.

The forward reachability of two tasks in the timed-path, according to the aforementioned functions ($att()$,$crit()$ and $wait()$), is examined based on the forward reachability function $forw()$ as presented in Eq. (9.17).

$$\begin{aligned}&forw(\tau_{abc}(w) \longrightarrow \tau_{dbe}(r)) = \\ &\neg att(\tau_{abc}(w) \longrightarrow \tau_{dbe}(r)) \wedge \\ &(\neg crit(\tau_{abc}(w) \longrightarrow \tau_{dbe}(r)) \vee wait(\tau_{abc}(w) \longrightarrow \tau_{dbe}(r)))\end{aligned} \tag{9.17}$$

Further, it is important to note that Eq. (9.17) does not cover all the cases to check the reachability of the timed-paths, because it can happen that two instances of a writer task reach to an instance of a reader task, e.g., when the period of the writer task is shorter than the reader task. In this case, the data that is accessible to the reader task will be overwritten by subsequent writer task instances. We make sure that only the last writer task instance reaches the reader task instance, i.e., there is no next writer task instance that reaches this reader task instance. This can be detected when the function in Eq. (9.18) returns true, where $\tau_{abc}(w+1)$ represents the next instance of the task instance $\tau_{abc}(w)$.

$$\begin{aligned}&reach(\tau_{abc}(w) \longrightarrow \tau_{dbe}(r)) = forw(\tau_{abc}(w) \longrightarrow \tau_{dbe}(r)) \\ &\wedge \neg forw(\tau_{abc}(w+1) \longrightarrow \tau_{dbe}(r))\end{aligned} \tag{9.18}$$

After checking the reachability between two task instances, we check for the whole timed-path by evaluating every two consecutive task instances in the timed-path from the first task until the last task. We show this by Eq. (9.19) where a timed-path tp_j^i belongs to the transaction Γ_j. For simplicity of the equation, two consecutive task instances in the timed-path are shown by τ_w and τ_r.

$$reach(tp_j^i) = \prod reach(\tau_w \longrightarrow \tau_r) \tag{9.19}$$

We finally evaluate all possible timed-paths in the transaction Γ_j to obtain all reachable (valid) timed-paths in a set TP_j^{reach}, according to Eq. (9.20) assuming that there are z timed-paths in the transaction.

$$TP_j^{reach} = \{reach(tp_j^i); i = 1..z\} \tag{9.20}$$

9.5.2 Accounting for the ST Messages

Each ST message has a deterministic schedule at each link within its route from the sender (writer) task to the receiver (reader) task. The activation time of an ST message on its last link (the time it is available to the reader task instance) is not directly dependent on the response time of the sender task or the response time of the message itself on the previous links. Although, some parameters such as the response time of the sender task, the transmission time of the ST message per link and/or other optimization objectives can be accounted for when defining the offset of the ST message per link. This is opposed to the case of a non-ST message, where the activation time of the message on the last link between the sender and receiver tasks is dependent upon both the response time of the sender task and its own response time on the previous links. Whereas, the ST messages are isolated by the time slots, which are configured offline on each link in their route to the receiver tasks.

ST offsets at the subsequent links in the route of the message are defined in a way to satisfy a set of scheduling constraints as presented in our previous work [52]. The scheduling constraints differ from the timing constraints which were mentioned earlier in this paper. The scheduling constraints are a set of logical rules set on the message's set of offsets per link to find satisfiable values of offsets which lead to a feasible offline schedule. The TSN schedules are configured offline, and before performing the data propagation delay analysis.

The most important set of scheduling constraints are as follows: 1) constraint on the frame size; 2) constraint on overlapping of messages on a link; 3) constraint on the order of traversed links; and 4) constraint on deadline satisfaction. Firstly, the constraint on the frame ensures that the value of the offset on one link does not force the arrival of the message after its next period activation. Secondly, the overlapping constraint checks whether the offset of two different messages on the same link will not cause overlapping time slots for these two messages. Thirdly, the order of the offset for the same message on subsequent links must consider the propagation of the message from the source end-station's link to the sink end-station's link (avoiding time-travel). Fourthly, the offsets per links in the route of the message must enforce the message to arrive to the destination end-station before the message's deadline (implicitly the next period activation of the sending task).

According to the feasible TSN schedule, the reachability of an instance of an ST message to the instances of the reader task is determined considering the offsets per link along the route of the message. In this case, the existing constraint to find reachable timed-path requires to trace through all the hops in the route of the ST message and take into account the activation times of the ST message per link.

An example of a transaction is shown in Figure 9.12, in which the ST class is utilized for the communication between two end-stations. The two end-stations are connected via the links l_1 and l_2, therefore the instance of the ST message at link l_1 is scheduled by the offset $O^{l_1}_{1,1}$ and it is notated by $m^{l_1}_{1,1}$. Likewise, the instance of the ST message on the link l_2 is indicated by the notation $m^{l_2}_{1,1}$ and its offset on the link is defined by $O^{l_2}_{1,1}$. Moreover, the reachable timed-paths in the transaction (Γ_1) shown in this example are subsequently, tp^1_1, tp^2_1, tp^3_1 and tp^4_1. For instance, tp^1_1 starts with the first instances of the first task ($\tau_{1,1,1}$) and the second task ($\tau_{1,1,2}$) of the source end-station. Then the message uses the bandwidth of the links in its path according to its offset at each of the links.

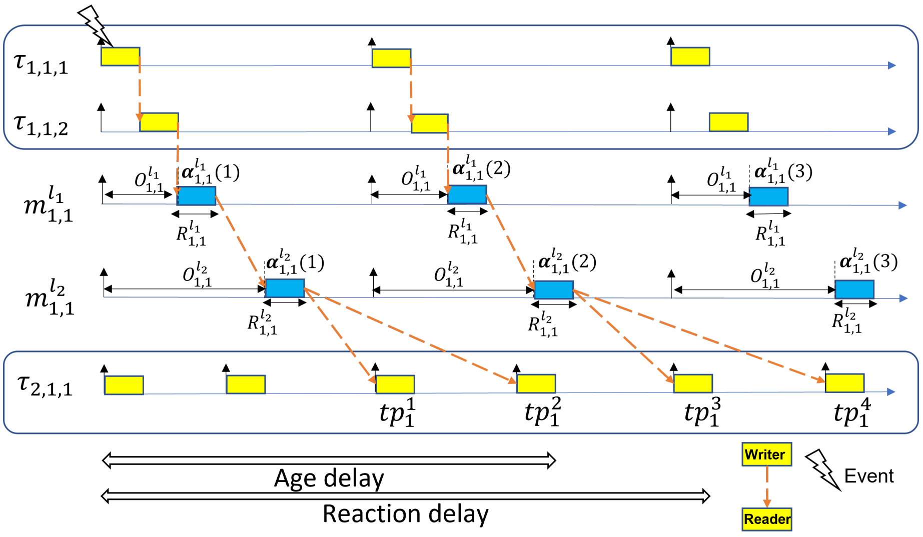


Figure 9.12. Timed-path with ST messages.

As it can be seen in Figure 9.12, the analysis has to be extended to support multiple activations of the message on several links as the ST messages have deterministic activation times per link. In the following, we show that in the end-to-end data propagation delay analysis of the ST messages, we can omit the activation times of the links in the path of the message except for the last link. This reduction in the timed-path significantly decreases the number of timed-paths to evaluate and in turn reduces the computation time of the analysis. We show this with the following lemma.

It is enough to consider the activation time of an ST message on its last link between its sender and receiver end-stations when extracting the reachable timed-paths through the TSN network.

Proof. Response-time analysis for ST traffic is not required as the ST traffic is scheduled offline. Feasible TSN schedules for ST class guarantee by construction the schedulability of the ST traffic. In addition, a feasible TSN schedule ensures that during the transmission of the ST message until its arrival to the destination end-station, there will be no new activation of the subsequent instances of the ST message. The activations and arrivals of ST frames within the network are known prior to the end-to-end analysis, thus there is always a deterministic reachable timed-path for an ST message over several links, which is already calculated and is configured offline, at the time of scheduling the ST traffic. Because the reachability of this deterministic path is already approved with the offline scheduling algorithm, it is only required to check the reachability of this path to the receiver node. Therefore, it is enough to consider the activation of the ST message on the last link (the end of the network's reachable path) when extracting the reachable timed-paths.

□

Based on Lemma 9.5.2 in case of the scenario depicted in Figure 9.12, the message at its last hop ($m^{l_2}_{1,1}$) is enough to consider in the identification of reachable timed-paths for the end-to-end data propagation delay analysis, which is shown in Figure 9.13. Hence, the existing analysis needs to be extended in order to support the TSN classes.

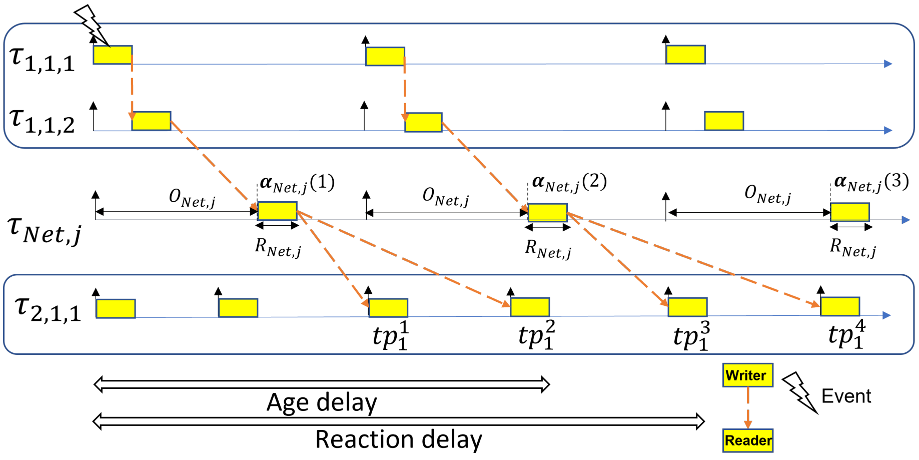


Figure 9.13. modeling the ST message at the last hop.

According to Lemma 9.5.2, only considering the ST message activation on its last link is sufficient for deriving the timed-path. However to consider the ST message on the last link, we propose to model the whole path of the ST message as a separate task which simplifies the incorporation of the ST message in the existing analysis. Eq. (9.21) shows the model of such a task. We regard this task as the network task and denote it by $\tau_{Net,j}$. This task corresponds to the message m^r_{jk}. The parameter r is the ID of the link delivering the message to the receiving end-station (last hop). Therefore, each ST message is modeled with a $\tau_{Net,j}$ task.

$$\tau_{Net,j} := \left\langle P_{Net,j} = P_{jk}, C_{Net,j} = TT(Size_{jk}), T_{Net,j} = T_{jk}, J_{Net,j} = J_{jk}, O_{Net,j} = O^{r}_{jk} \right\rangle \tag{9.21}$$

where, $TT()$ calculates the transmission time of the message based on the size of the message and the network speed (s). $TT()$ is calculated by $((Size_{jk} + OH) * 8)/(s)$. Where, OH is the TSN frame's overhead in Bytes.

Modeling of the ST message with a task allows us to use Eq. (9.20) to evaluate the reachability of timed-paths corresponding to a transaction, where $\tau_{Net,j}$ can be a reader or writer task in the transaction j, while its activation time is computed according to Eq. (9.10).

9.5.3 Accounting for the Non-ST Messages

A non-ST message is assumed to be scheduled for transmission as soon as the sender task completes its execution. The arrival time of the message instance to the reader task instance is calculated based on the activation time and the period of the writer task instance and the response time of the message. As depicted in Figure 9.14, the non-ST message $m_{1,1}$ has no offset in the set of two links in its route to the reader task, namely $\tau_{2,1,1}$. As a result, the activation time of the message is assumed to be the same as its predecessor writer task. In Figure 9.14, each instance of $m_{1,1}$ is not transmitted immediately after the completion of the sender task $\tau_{1,1,2}$ because we assume the message received interference from other higher priority messages (and/or it was blocked by the lower priority messages). By knowing the worst-case response time of the message $(R_{1,1})$ the reachable instance of the reader task can be determined. For example, consider tp_1^1 and tp_1^2 in Figure 9.14. In these timed-paths, the first instance of the message is reachable to the third and fourth instance of the reader task. Therefore, Eq. (9.17) is used to evaluate the reachability of timed-paths in a transaction.

9.5.4 Calculation of Worst-case Data Age and Reaction Time Delays

The data age and reaction time delays are derived based on timed-paths (introduced in Section 9.5.1). In this section, we further explain the calculation of the worst-case data age and reaction time delays according to [5]. The data age delay for a timed-path tp_j^n belonging to the transaction Γ_j is calculated by Eq. (9.22), which calculates the time difference between the input data and the last sample of the corresponding output data.

$$\Delta_{age}(tp_j^n) = \alpha_{last}(tp_j^n) + RT_{last}(tp_j^n) - \alpha_{first}(tp_j^n) \tag{9.22}$$

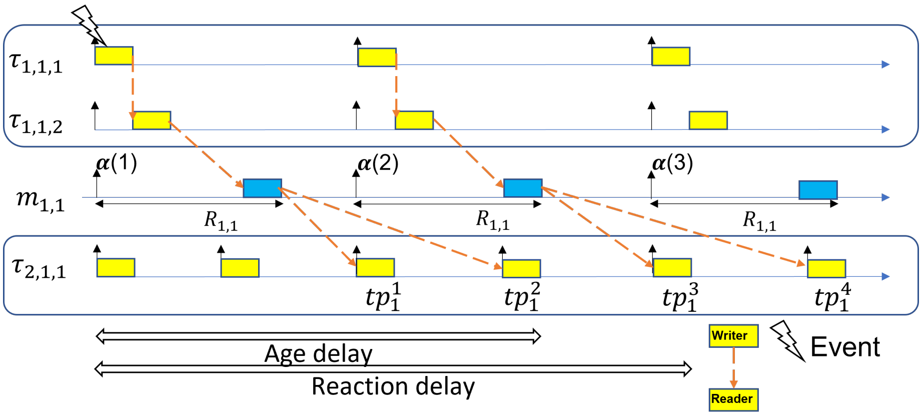


Figure 9.14. Timed-path with non-ST messages.

where, $\alpha_{first}()$ returns the activation time of the task instance that is the first task receiving the fresh input data. This task is an instance of the transaction's initiator task inside the initiator end-station. Also, $\alpha_{last}()$ and $RT_{last}()$ return the activation time and the worst-case response time of the instance of the terminator task from the terminator end-station, after which the data is overwritten.

The reaction time delay is calculated by Eq. (9.23), where $Pred()$ represents the first instance of the timed-path before the timed-path under analysis tp_j^n. Note that the effect of just missing an event at the input of the task chain is covered by $Pred()$.

$$\Delta_{reac}(tp_j^n) = \alpha_{last}(tp_j^n) + RT_{last}(tp_j^n) - \alpha_{first}(Pred(tp_j^n)) \tag{9.23}$$

The data age and reaction time delays for all timed-paths for every transaction should be extracted and the longest corresponding values represent the worst-case data age and reaction delays, which are calculated according to Eq. (9.24).

$$\begin{aligned} \Delta_{age}(\Gamma_j) &= \{max(\Delta_{age}(tp_j^n)); n = 1...z\} \\ \Delta_{reac}(\Gamma_j) &= \{max(\Delta_{reac}(tp_j^n)); n = 1...z\} \end{aligned} \tag{9.24}$$

The instances of the task set in a periodic system repeat in a given bounded pattern specified by the LCM of the periods of the tasks in the task set. Moreover, to find the worst-case data age and reaction time delays, it is sufficient to enumerate and compare all timed-paths within a finite bound of twice the hyperperiod (LCM of all the involved periods in the transaction) [5].

It is conventionally desired that the data age and reaction time delays are less than or equal to their corresponding constraints according to Eq. (9.25).

$$\begin{aligned}\Delta_{age}(\Gamma_j) &<= Age_j\\ \Delta_{reac}(\Gamma_j) &<= Reac_j\end{aligned} \tag{9.25}$$

After the analysis the values of the data age and reaction time delays can be examined to check if they satisfy the specific needs of the targeted system according to the user-defined end-to-end timing constraints, i.e., based on different criteria as explained in the works [16, 48].

9.6 Vehicular Application Case Study

In this section, we discuss a vehicular industrial use case that is used to evaluate the presented end-to-end data propagation delay analysis. The use case consists of 14 end-stations that are connected by a two-switch TSN network as illustrated in Figure 9.15. Each end-station is assumed to include multiple tasks. Figure 9.15 is inspired by a use case developed in [53] and the traffic is specified accordingly. We implemented the proposed analysis as an in-house tool. The configuration and message set in the use case were given as input to the implemented analysis.

9.6.1 Experimental Setup

We perform the extended as well as the existing end-to-end data propagation delay analysis of the use case in Figure 9.15. The evaluation of the analyses is performed under different traffic scenarios given to the use case. In this experiment, we assume there are 14 transactions starting from end-stations 1 to 7 that use different TSN traffic classes to communicate with three sink end-stations with the IDs 8 to 10. Table 9.1 shows the transaction settings. Moreover, Figure 9.16 illustrates the relation between the elements of the transactions shown in Table 9.1. Note that in Figure 9.16, we do not show registers between any two messages within a node or two end-stations for the sake of enhancing readability of the figure. Each transaction initiates and terminates by a task within different end-stations. Each transaction includes four tasks in total. Besides, each transaction includes two tasks per end-station which constitute the chain. In the source end-station, the first task is a computation task and the subsequent task is a communication task, which receives data input from the predecessor task, then prepares and injects messages to the network. In the destination end-station of these transactions,

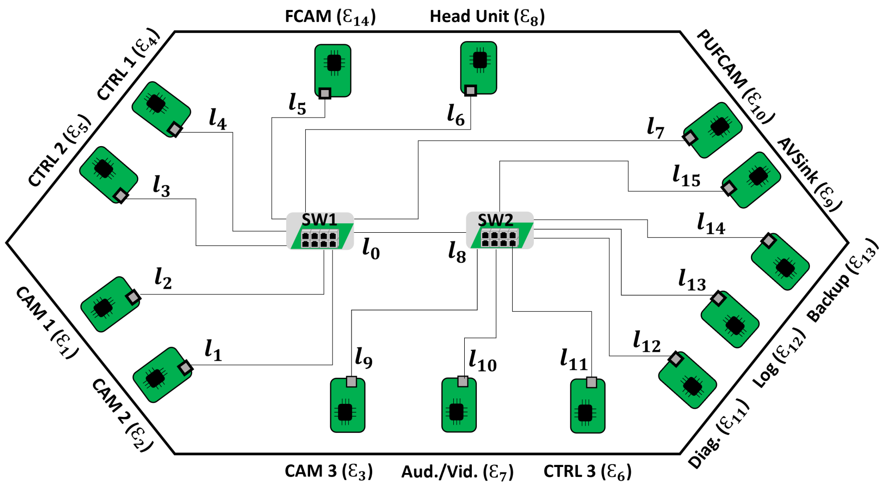


Figure 9.15. Vehicular application use case.

the first task is a communication task that receives and processes the message from the network. The communication task sends the message for further processing to the last task in the transaction. The periods of the tasks are chosen based on the automotive data set in [54], i.e., $\{1, 2, 5, 10, 20, 50, 100\}$ in milliseconds (ms). The size of all messages are fixed to 1500 $Bytes$ as the maximum payload size and the worst-case execution time of each task (WCET) is considered to be 0.5 ms. The end-stations run their tasks according to the fixed-priority preemptive scheduling algorithm. In each transaction, we assume that the priority of a task is higher than the priority of its subsequent task within the same end-station. Because, we want to keep the execution of two subsequent tasks inside the same end-station in such an order to avoid creating delay in writing of data. This ensures that each precedent task (writer task) in the transaction must be executed before the subsequent task in the chain (reader task) in the case if the two tasks are activated for execution at the same time.

Five out of fourteen transactions use ST traffic class; three transactions use class A; three transactions use class B; and three transactions use class BE. The CBS mechanism is set according to Table 9.2, where the credit for classes A and B are chosen according to the utilization of these classes on the network links. The overall network speed is set to 1 $Gbps$. We set the idle slope (*idleSlope*) according to the utilization of the traffic on classes A and B, as shown in Table 9.2. The transactions 6, 7 and 8 use class A on the links 10 and 15, therefore the credit of the links l_{10} and l_{15} are set to 0.78. Furthermore,

Table 9.1. Evaluation settings for the use case based on distributed chains.

Γ_j	$\mathcal{E}_i$	Source tasks (τ_{ijk}): [id, P_{ijk},C_{ijk}, T_{ijk}]		Message (m_{jk}): [id, P_{jk},$Size_{jk}$, T_{jk}, $\mathcal{O}^r_{jk}$]	$\mathcal{E}_i$	Destination tasks (τ_{ijk}): [id, P_{ijk},C_{ijk}, T_{ijk}]	
		Computation	Communication			Communication	Computation
1	$\mathcal{E}_1$	[$\tau_{1,1,1}$,4,0.5,20]	[$\tau_{1,1,2}$,3,0.5,20]	[$m_{1,1}$, ST,1500,20,1.039]	$\mathcal{E}_8$	[$\tau_{8,1,1}$,10,0.5,10]	[$\tau_{8,1,2}$,9,0.5,10]
2	$\mathcal{E}_3$	[$\tau_{3,2,1}$,4,0.5,20]	[$\tau_{3,2,2}$,3,0.5,20]	[$m_{2,2}$, ST,1500,20,1.026]	$\mathcal{E}_8$	[$\tau_{8,2,3}$,8,0.5,10]	[$\tau_{8,2,4}$,7,0.5,10]
3	$\mathcal{E}_4$	[$\tau_{4,3,1}$,4,0.5,10]	[$\tau_{4,3,2}$,3,0.5,10]	[$m_{3,3}$, ST,1500,10,1.065]	$\mathcal{E}_8$	[$\tau_{8,3,5}$,6,0.5,10]	[$\tau_{8,3,6}$,5,0.5,10]
4	$\mathcal{E}_6$	[$\tau_{6,4,1}$,4,0.5,10]	[$\tau_{6,4,2}$,3,0.5,10]	[$m_{4,4}$, ST,1500,10,1.078]	$\mathcal{E}_8$	[$\tau_{8,4,7}$,4,0.5,10]	[$\tau_{8,4,8}$,3,0.5,10]
5	$\mathcal{E}_5$	[$\tau_{5,5,1}$,2,0.5,10]	[$\tau_{5,5,2}$,1,0.5,10]	[$m_{5,5}$, ST,1500,10,1.052]	$\mathcal{E}_9$	[$\tau_{9,5,1}$,10,0.5,10]	[$\tau_{9,5,2}$,9,0.5,10]
6	$\mathcal{E}_7$	[$\tau_{7,6,1}$,6,0.5,10]	[$\tau_{7,6,2}$,5,0.5,10]	[$m_{6,6}$, A,1500,10,0]	$\mathcal{E}_9$	[$\tau_{9,6,3}$,8,0.5,10]	[$\tau_{9,6,4}$,7,0.5,10]
7	$\mathcal{E}_7$	[$\tau_{7,7,3}$,4,0.5,10]	[$\tau_{7,7,4}$,3,0.5,10]	[$m_{7,7}$, A,1500,10,0]	$\mathcal{E}_9$	[$\tau_{9,7,5}$,6,0.5,10]	[$\tau_{9,7,6}$,5,0.5,10]
8	$\mathcal{E}_7$	[$\tau_{7,8,5}$,2,0.5,10]	[$\tau_{7,8,6}$,1,0.5,10]	[$m_{8,8}$, A,1500,10,0]	$\mathcal{E}_9$	[$\tau_{9,8,7}$,4,0.5,10]	[$\tau_{9,8,8}$,3,0.5,10]
9	$\mathcal{E}_1$	[$\tau_{1,9,3}$,2,0.5,20]	[$\tau_{1,9,4}$,1,0.5,20]	[$m_{9,9}$, B,1500,20,0]	$\mathcal{E}_{10}$	[$\tau_{10,9,1}$,8,0.5,10]	[$\tau_{10,9,2}$,7,0.5,10]
10	$\mathcal{E}_2$	[$\tau_{2,10,1}$,4,0.5,20]	[$\tau_{2,10,2}$,3,0.5,20]	[$m_{10,10}$, B,1500,20,0]	$\mathcal{E}_{10}$	[$\tau_{10,10,3}$,6,0.5,10]	[$\tau_{10,10,4}$,5,0.5,10]
11	$\mathcal{E}_2$	[$\tau_{2,11,3}$,2,0.5,20]	[$\tau_{2,11,4}$,1,0.5,20]	[$m_{11,11}$, B,1500,20,0]	$\mathcal{E}_{10}$	[$\tau_{10,11,5}$,4,0.5,10]	[$\tau_{10,11,6}$,3,0.5,10]
12	$\mathcal{E}_3$	[$\tau_{3,12,3}$,2,0.5,20]	[$\tau_{3,12,4}$,1,0.5,20]	[$m_{12,12}$, BE,1500,20,0]	$\mathcal{E}_8$	[$\tau_{8,12,9}$,2,0.5,10]	[$\tau_{8,12,10}$,1,0.5,10]
13	$\mathcal{E}_4$	[$\tau_{4,13,3}$,2,0.5,10]	[$\tau_{4,13,4}$,1,0.5,10]	[$m_{13,13}$, BE,1500,10,0]	$\mathcal{E}_{10}$	[$\tau_{10,13,7}$,2,0.5,10]	[$\tau_{10,13,8}$,1,0.5,10]
14	$\mathcal{E}_6$	[$\tau_{6,14,3}$,2,0.5,10]	[$\tau_{6,14,4}$,1,0.5,10]	[$m_{14,14}$, BE,1500,10,0]	$\mathcal{E}_9$	[$\tau_{9,14,9}$,2,0.5,10]	[$\tau_{9,14,10}$,1,0.5,10]

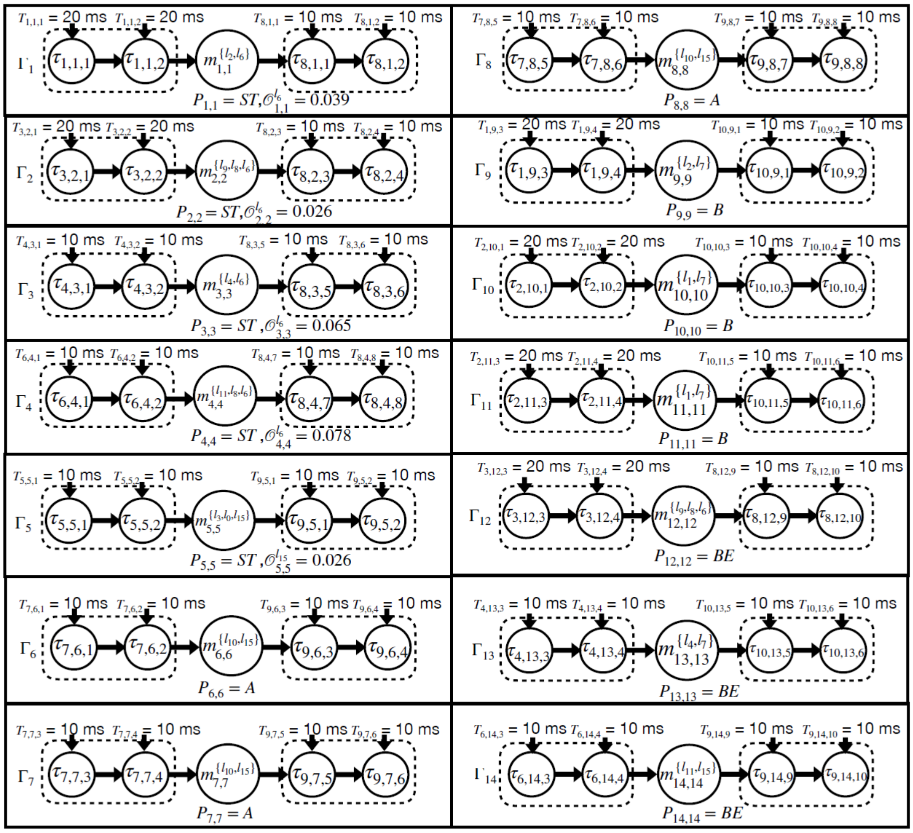


Figure 9.16. Evaluation settings (chains of tasks).

the transactions 9, 10 and 11 use the class B on the links 1, 2 and 7. Accordingly, the credits for class B on the links l_1, l_2 and l_7 are set to 0.4. We note that zero credit means there are no messages from the associated CBS classes on the link. In Table 9.2 only the credit for the links utilizing CBS is shown. Finally, the data age and reaction time constraints specified on each transaction are depicted in Table 9.3.

Table 9.2. idleSlope of class A and class B per link.

idleSlope ($Mbps$)	l_1	l_2	l_7	l_{10}	l_{15}
Class A	-	-	-	0.78	0.78
Class B	0.4	0.4	0.4	-	-

Table 9.3. Timing constraints for all transactions.

	Reac$_j$ (ms)	Age$_j$ (ms)
Cr$_j$	35	25

9.6.2 Evaluation of the Existing and Extended Analyses

In this section, we compare the analysis results acquired by performing the existing [5] and the extended end-to-end data propagation delay analyses. Table 9.4 shows the response times of each message ($R_{i,j}$) and its sending task ($R_{i,j,k}$) separately. Moreover, the response time of the last task in each transaction is shown in Table 9.4.

Table 9.4. Response times of the tasks and messages in the transactions (ms).

Trans. (Γ_j)	R_{ijk} of sender	R_{ij} of message	R_{ijk} of last task
1	$R_{1,1,2} = 1$	$R_{1,1} = 0.025$	$R_{8,1,2} = 1$
2	$R_{3,2,2} = 1$	$R_{2,2} = 0.038$	$R_{8,2,4} = 2$
3	$R_{4,3,2} = 1$	$R_{3,3} = 0.025$	$R_{8,3,6} = 3$
4	$R_{6,4,2} = 1$	$R_{4,4} = 0.038$	$R_{8,4,8} = 4$
5	$R_{5,5,2} = 1$	$R_{5,5} = 0.038$	$R_{9,5,2} = 1$
6	$R_{7,6,2} = 1$	$R_{6,6} = 1.081$	$R_{9,6,4} = 2$
7	$R_{7,7,4} = 2$	$R_{7,7} = 2.081$	$R_{9,7,6} = 3$
8	$R_{7,8,6} = 3$	$R_{8,8} = 3.081$	$R_{9,8,8} = 4$
9	$R_{1,9,4} = 2$	$R_{9,9} = 2.218$	$R_{10,9,2} = 1$
10	$R_{2,10,2} = 1$	$R_{10,10} = 1.262$	$R_{10,10,4} = 2$
11	$R_{2,11,4} = 2$	$R_{11,11} = 2.262$	$R_{10,11,6} = 3$
12	$R_{3,12,4} = 2$	$R_{12,12} = 2.127$	$R_{8,12,10} = 5$
13	$R_{4,13,4} = 2$	$R_{13,13} = 3.150$	$R_{10,13,8} = 4$
14	$R_{6,14,4} = 2$	$R_{14,14} = 2.398$	$R_{9,14,10} = 5$

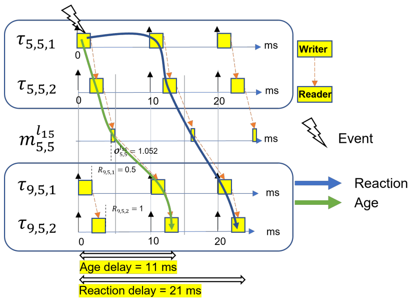
Table 9.5. Calculated reaction time and data age delays for each transaction.

	Results with the existing analysis		Results with the extended analysis	
Trans. (Γ_j)	Reaction Delay (ms)	Age Delay (ms)	Reaction Delay (ms)	Age Delay (ms)
1	32.064	22.064	31	21
2	33.064	23.064	32	22
3	24.09	14.09	23	13
4	25.116	15.116	24	14
5	22.09	12.09	21	11
6	23.081	13.081	22	12
7	25.081	15.081	23	13
8	27.081	17.081	24	14
9	33.218	23.218	31	21
10	33.262	23.262	32	22
11	35.262	25.262	33	23
12	37.127	27.127	35	25
13	27.15	17.15	24	14
14	27.398	17.398	25	15

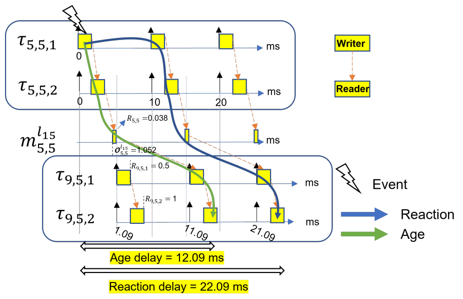
The data age and reaction time delays for each individual transaction depicted in Table 9.1 are calculated with the existing and extended end-to-end data-propagation delay analyses. The analyses results are shown in Table 9.5.

It can be observed in Table 9.5 that the reaction time and data age delays calculated by the extended end-to-end data-propagation delay analysis are smaller than those calculated by the existing analysis. For example, the reaction time and data age delays of transaction Γ_5 calculated with the existing analysis are 22.09 ms and 12.09 ms respectively. Whereas, the reaction time and data age delays of transaction Γ_5 calculated with the extended analysis are 21 ms and 11 ms respectively. We note that the existing analysis calculates the reaction time and data age delays of Γ_5 with 5.19$\%$ and 9.90$\%$ over-estimation (pessimism) compared to the extended analysis. We visually demonstrate the reaction time and data age delays in Γ_5 with the help of execution traces in Figure 9.17. In Γ_5, the tasks in the sender end-station ($\mathcal{E}_5$) and the receiver end-station ($\mathcal{E}_9$) are synchronized for using the ST class. The extended analysis considers synchronization of the nodes in Γ_5, i.e., each node sees the same time 0. Figure 9.17(a) shows the reachable timed-paths within twice the hyperperiod (LCM of the periods of all tasks in Γ_5, which is 20 ms). On the other hand, the execution trace considered by the existing analysis is shown in Figure 9.17(b).

Similarly, consider another transaction, Γ_{11}, in Table 9.5. The reaction time and data age delays of transaction Γ_{11} calculated with the existing analysis are subsequently 35.262 ms and 25.262 ms. Whereas, these delays calculated with the extended analysis are 33 ms and 23 ms respectively. Hence, the existing analysis calculates the reaction



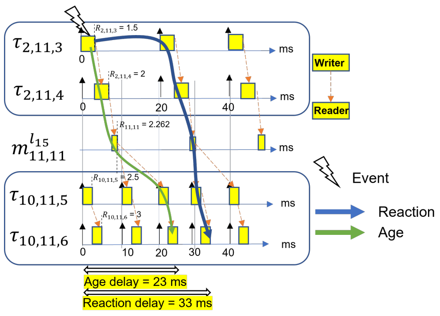
(a) Synchronized end-stations.



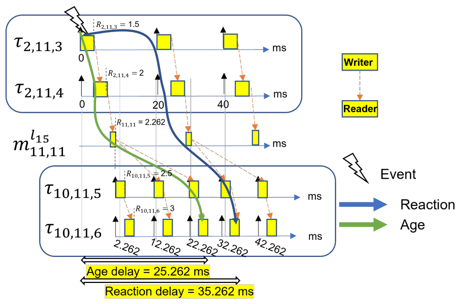
(b) Non-synchronized end-stations.

Figure 9.17. Demonstration of data age and reaction time delays in Γ_5 in Table 9.5 with with execution traces.

time and data age delays of Γ_{11} with 6.85% and 9.83% over-estimation (pessimism) compared to the extended analysis. The reaction time and data age delays in Γ_{11} are visually demonstrated with the help of execution traces in Figure 9.18. Γ_{11} uses a class B message. The extended analysis considers synchronization of the nodes in Γ_{11}, i.e., each node sees the same time 0 as shown in the execution trace in Figure 9.18(a). Whereas, the execution trace considered by the existing analysis is shown in Figure 9.18(b).



(a) Synchronized end-stations.



(b) Non-synchronized end-stations.

Figure 9.18. Demonstration of data age and reaction time delays in Γ_{11} in Table 9.5 with with execution traces.

9.6.3 Impact of Various Parameters on the Data Age and Reaction Time Delays

In this subsection, we demonstrate the impact of various parameters on the data age and reaction time delays using the extended end-to-end data propagation delay analysis. The parameters of interest include the offset of the ST messages, and the periods of the tasks in the receiver end-stations (i.e., the periods of the message receiving tasks). The evaluations are carried out in a transaction in the use case shown in Figure 9.15. The transaction's details are shown in Table 9.6.

Table 9.6. The transaction specifications.

Γ_d	$\mathcal{E}_i$	Source tasks (τ_{ijk}): [id, P_{ijk},C_{ijk}, T_{ijk}]	Message (m_{jk}): [id, P_{jk},$Size_{jk}$, T_{jk}, $\mathcal{O}^r_{jk}$]	$\mathcal{E}_i$	Destination tasks (τ_{ijk}): [id, P_{ijk},C_{ijk}, T_{ijk}]	
		Task 1			Task 2	Task 3
1	$\mathcal{E}_3$	[$\tau_{1,1,1}$,1,1,50]	[$m_{1,1}$, ST,1500,50,X]	$\mathcal{E}_8$	[$\tau_{9,1,1}$,2,1,Y]	[$\tau_{9,1,2}$,1,1,Y]

The transaction under analysis starts from CAM1 as the sending end-station and finishes in AVSink as the receiving end-station. This transaction uses one task in CAM1 with the fixed period of 50 ms and the WCET of 1 ms. CAM1 sends an ST message that is routed to AVSink via three links, namely l_2, l_0 and l_{15} according to the topology shown in Figure 9.15. There are two tasks in the AVSink that are engaged in this transaction, namely $\tau_{9,1,1}$ and $\tau_{9,1,2}$. Each of these tasks has the WCET of 1 ms. The task $\tau_{9,1,1}$ in AVSink that receives messages from the TSN network has the highest priority. We vary the period of the receiver end-station's tasks (AVSink) shown by notation Y in Table 9.6. Y can obtain values from the range $\{15, 20, 25, 30, 40, 50, 60, 70, 90, 100, 150, 200, 250\}$ in milliseconds (ms). Both tasks of AVSink have the same period.

As we consider an ST message in the transaction under analysis, we can assign any value to its offset between the range of *minimum offset* and *maximum offset* on its last link $m^{l_{15}}_{1,1}$. The variable value of the offset is denoted by X notation in Table 9.6. The minimum offset of the message on the last link ($m^{l_{15}}_{1,1}$) is equal to the sum of the response time of the sender task, $\tau_{1,1,1}$ (1 ms), transmission time of the message on the first link $m^{l_2}_{1,1}$ (0.0126 ms) and the transmission time of the message on the second link $m^{l_0}_{1,1}$ (0.0126 ms) as shown in Figure 9.19. The sum of these three terms is equal to 1.0252 ms, which defines the minimum value of the offset that can be assigned to the message under analysis on its last link towards the destination end-station. Similarly, the maximum value of the offset that can be assigned to the message under analysis on its last link towards the destination end-station is 49.9874 ms, which is equal to the

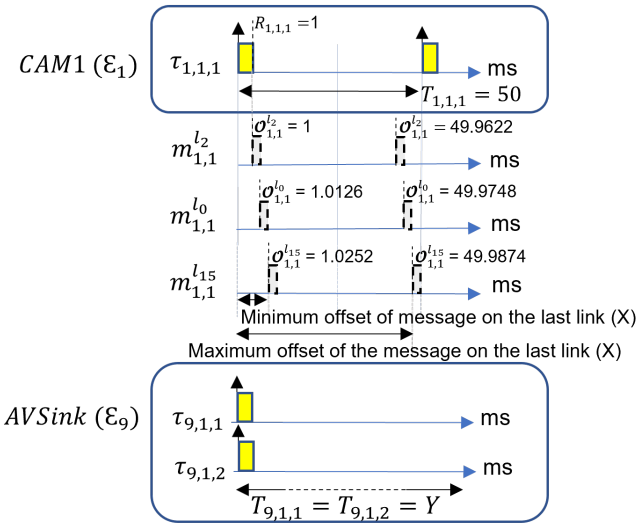

$CAM1$ ($\mathcal{E}_1$)
$\tau_{1,1,1}$
$R_{1,1,1}$ =1
ms
$T_{1,1,1}$ = 50
$m^{l_2}_{1,1}$
$O^{l_2}_{1,1}$ = 1
$O^{l_2}_{1,1}$ = 49.9622
ms
$m^{l_0}_{1,1}$
$O^{l_0}_{1,1}$ = 1.0126
$O^{l_0}_{1,1}$ = 49.9748
ms
$m^{l_{15}}_{1,1}$
$O^{l_{15}}_{1,1}$ = 1.0252
$O^{l_{15}}_{1,1}$ = 49.9874
ms
Minimum offset of message on the last link (X)
Maximum offset of the message on the last link (X)
$AVSink$ ($\mathcal{E}_9$)
$\tau_{9,1,1}$
ms
$\tau_{9,1,2}$
ms
$T_{9,1,1} = T_{9,1,2} = Y$

Figure 9.19. Demonstration of the minimum and maximum values of the offsets on the last link for the message under analysis.

difference between the message's period (50 ms) and transmission time of the message on the last link (0.0126 ms). There are no other TSN messages interfering with the ST message in this transaction. According to the Worst-Case Response-Time Analysis of TSN [34, 33, 31], the response time of the message (the time the message is received at the receiver end-station) is 1.025 ms. The experiments are carried out by calculating and comparing the data age and reaction time delays, under variations made on the parameters X for ST offsets and Y for the period of the receiver end-station's tasks from the settings given in Table 9.6.

The data age and reaction time delays of the transaction under analysis are presented in Figure 9.20 and Figure 9.21 respectively. The horizontal axes in Figure 9.20 and Figure 9.21 show the variations in the receiver task's periods. The vertical axis represents the data age delays (in Figure 9.20) and reaction time delays (in Figure 9.21). Color-coded line graphs present the delays calculated based on different offsets assigned to the ST message.

Figure 9.20 shows that the data age delay remains constant at 52 ms with variation in the period of the message receiving task in the transaction, when the offset of the message is set to the minimum value of 1.0252 ms. The data age delay increases as the value of the offset is increased to its maximum value of 49.9874 ms. Similarly, the reaction time delays calculated for the same transaction by setting different values of the ST message offset and by varying the period of the message receiving task within the traction are depicted in Figure 9.21.

We note for some specific cases in Figure 9.20, where the message receiving task's period is a harmonic multiple (50 ms, 100 ms, ...) of the sender task's period (50 ms) within the transaction, the data age delay remain constant at 52 ms. Furthermore, any variation in the ST message's offset (selected between the minimum and maximum values of the offset) also does not effect the data age delay in such cases. The reason for getting the same data age delay in these specific cases is that the data age delay considers the last input in the timed path which is not overwritten by the previous inputs [5, 7]. We demonstrate this situation by drawing the execution traces of the transaction in Figure 9.22 and Figure 9.23, where the period of the message receiving task ($\tau_{9,1,1}$) is set to the first two harmonic multiples (50 ms and 100 ms) of the period of the sender task ($\tau_{1,1,1}$, period = 50 ms). Figure 9.22(a, c, e) and Figure 9.23(a, c, e) represent the execution traces of the transaction when periods of the sender task ($\tau_{1,1,1}$) and the receiver task ($\tau_{9,1,1}$) are set to 50 ms while different values of the ST message offset are considered. Whereas, Figure 9.22(b, d, f) and Figure 9.23 (b, d, f) represent the execution traces of the transaction when periods of the sender task ($\tau_{1,1,1}$) is set to 50 ms while the period of the receiver task ($\tau_{9,1,1}$) is set to 100 ms while varying the ST message's offset.

For instance, the data age delay in Figure 9.22(a) is 52 ms. Although the period of ($\tau_{9,1,1}$) is doubled in Figure 9.22(b) with respect to its period in Figure 9.22(a), the first instance of the message will be overwritten by the second instance of the message in Figure 9.22(b). Hence, the data provided by the first instance of $\tau_{1,1,1}$ via the first instance of the message cannot reach the second instance of $\tau_{9,1,1}$. In fact, the second instance of $\tau_{9,1,1}$ will read the data produced by the second instance of $\tau_{1,1,1}$. Therefore, the data age delay in Figure 9.22(b) is the same as that of the data age delay in Figure 9.22(a). In fact, the data age delay stays the same when higher harmonic multiples of the period of $\tau_{1,1,1}$ are considered for the period of $\tau_{9,1,1}$ including 150 ms, 200 ms and 250 ms as shown in Figure 9.20. Furthermore, by varying the values of the ST message's offsets, the data path for the age delay remains the same because the message offset cannot exceed the period of its sender task ($\tau_{1,1,1}$) and the period of the receiver task ($\tau_{9,1,1}$) is a harmonic multiple of the period of $\tau_{1,1,1}$.

On the other hand, the reaction time delay is significantly impacted by the increase in the period of the task that is receiving the message, as shown in Figure 9.21. We note that the reaction time delay is the same for the periods of $\tau_{9,1,1}$ that are the first two harmonic multiples of the period of $\tau_{1,1,1}$. However, the reaction delay keeps on increasing with the increase in the period of $\tau_{9,1,1}$ that is equal to the subsequent harmonic multiples (150 ms, 200 ms and 250 ms) of the period of $\tau_{1,1,1}$ as shown in

Figure 9.21.

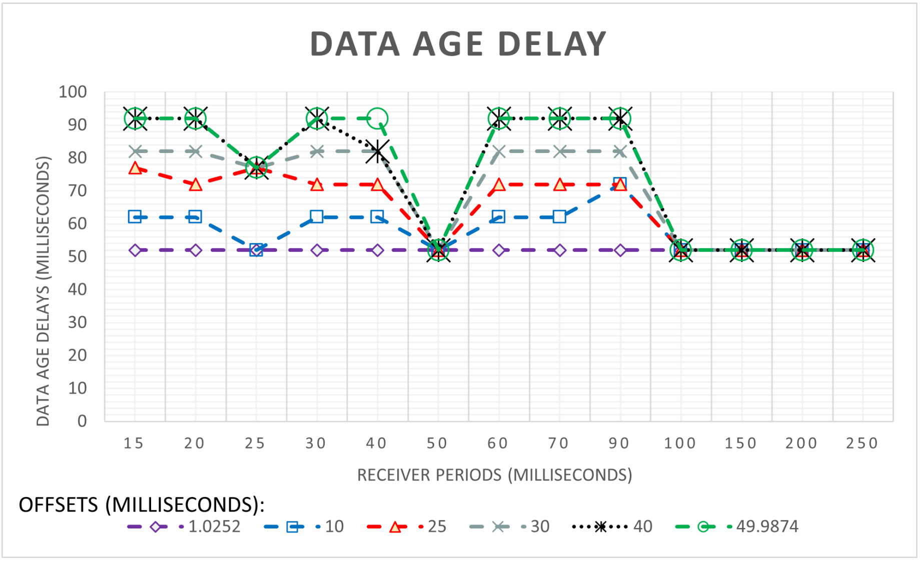


Figure 9.20. Impact of variations on the message offsets and the receiver task's period on the data age delay.

9.6.4 Discussions

The end-to-end data propagation delays such as data age and reaction time delays calculated by the existing timing analysis can be pessimistic (over-estimated) based on different network implementations, i.e., when the sender and receiver end-stations are synchronized. Also, variations in the network configuration such as offset configuration or period of the sender and receiver tasks affect the propagation of data from the input to the output of the transaction. Hence the end-to-end data propagation delays may also vary significantly for different settings. Transactions that utilize ST class need more configuration and optimization effort due to the need for offline schedules. Though, ST classes enable flexible adjustment of the end-to-end data-propagation delays.

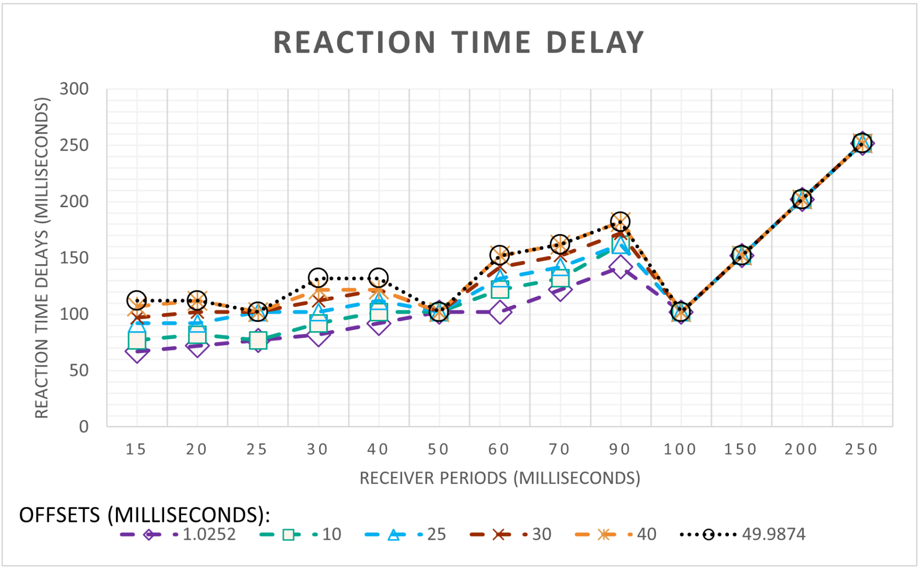


Figure 9.21. Impact of variations on the message offsets and the receiver task's period on the reaction time delay.

9.7 Conclusions and Future Works

In this paper, we presented a detailed end-to-end timing model of vehicular distributed embedded systems that utilize TSN for network communication. The end-to-end timing model enables integrating TSN features with the existing end-to-end data propagation analysis. The proposed end-to-end timing model features distributed TSN transactions over two or multiple synchronized end-stations with locally synchronized tasks. Distributed TSN transactions define a chain of tasks which can communicate within two different end-stations by TSN messages. TSN classes have different clock synchronization requirements, that have not been accounted for in the existing end-to-end data propagation delay analysis. In this paper, we proposed models and methods to incorporate all TSN traffic requirements in the end-to-end data propagation delay analysis.

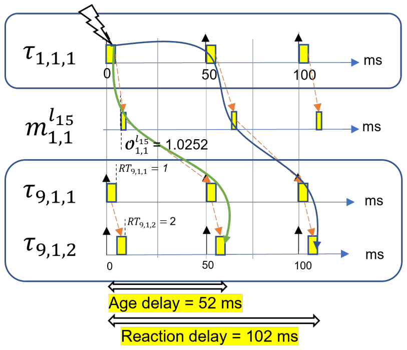
The proposed end-to-end timing model and analysis were evaluated on a vehicular use case. We performed a comparative evaluation of the existing and extended analysis on the base of the use case. Moreover, the evaluations are in terms of the effects that the transaction periods and the message offsets can cause on the data age and reaction time delays.

In summary, we concluded from the results that the non-ST transactions deal with less effective parameters on the data age and reaction time delays. The data age and reaction time delays of the ST transactions are more flexibly readjusted, though the optimization with regards to their multiple configurable parameters is complex.

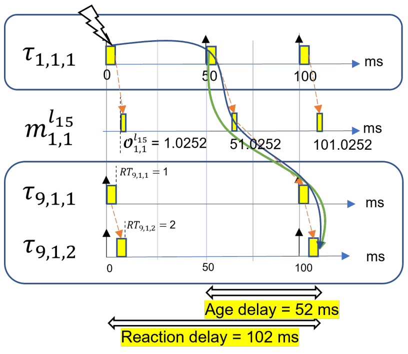
The results of this work additionally indicated some potential future work directions. Firstly, this work enables analysis-based optimization of the end-to-end timing delays (active approach). For example, the scheduling of the tasks within the end-stations could also be included in order to optimize the end-to-end data propagation delays. Secondly, system-level optimization of the end-to-end data propagation delays by co-scheduling of tasks and messages could be considered as extension of the work presented in this paper. Finally, a potential future work is to integrate the proposed end-to-end data propagation delay analysis with model-based software development frameworks for distributed embedded systems, e.g., Rubus-ICE [12].

Acknowledgements

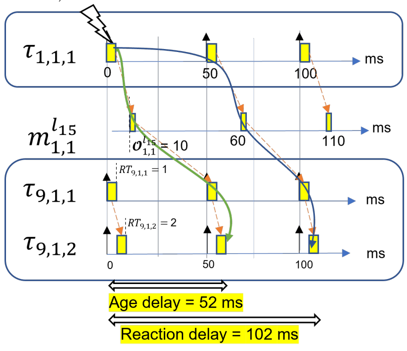
The work in this paper is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the DESTINE, PROVIDENT and INTERCONNECT projects, and the Swedish Knowledge Foundation via the FIESTA, HERO and DPAC projects. We would like to thank our all industrial partners, especially Volvo Construction Equipment, Arcticus Systems, and HIAB.



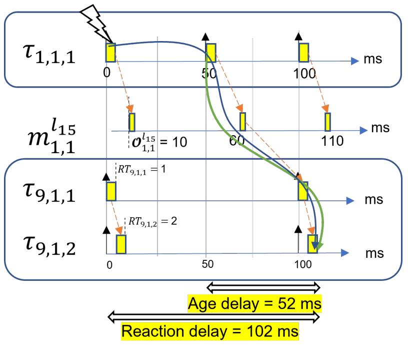
(a) $\mathcal{O}_{1,1}^{l_{15}}$ = 1.0252 ms; $T_{9,1,1} = T_{9,1,2}$ = 50 ms.



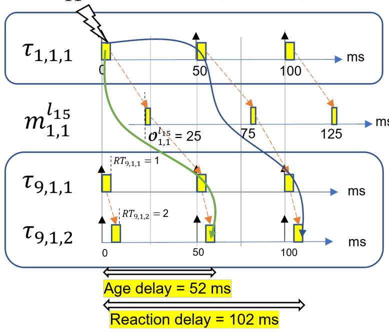
(b) $\mathcal{O}_{1,1}^{l_{15}}$ = 1.0252 ms;
$T_{9,1,1} = T_{9,1,2}$ = 100 ms.



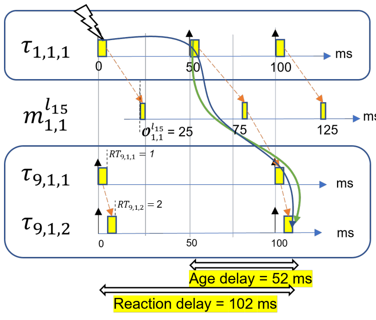
(c) $\mathcal{O}_{11}^{l_{15}}$ = 10 ms; $T_{9,1,1} = T_{9,1,2}$ = 50 ms.



(d) $\mathcal{O}_{11}^{l_{15}}$ = 10 ms; $T_{9,1,1} = T_{9,1,2}$ = 100 ms.

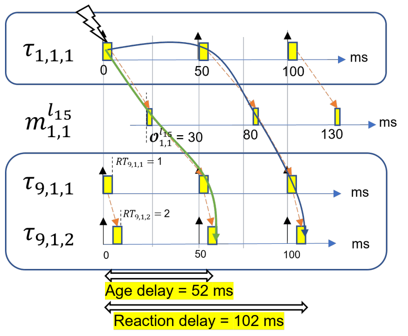


(e) $\mathcal{O}_{11}^{l_{15}}$ = 25 ms; $T_{9,1,1} = T_{9,1,2}$ = 50 ms.

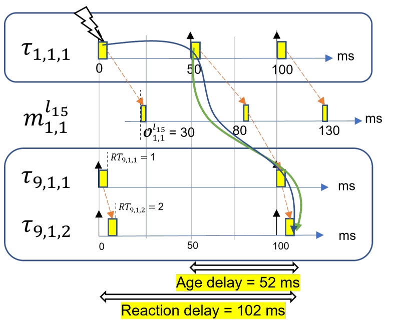


(f) $\mathcal{O}_{11}^{l_{15}}$ = 25 ms; $T_{9,1,1} = T_{9,1,2}$ = 100 ms.

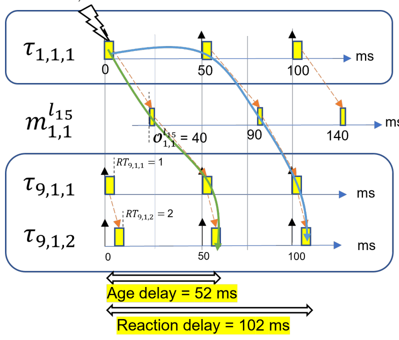
Figure 9.22. The effect of offset variations and harmonically set periods on the data age and reaction time delays of the transaction (offsets 1,012 ms, 10 ms and 25 ms).



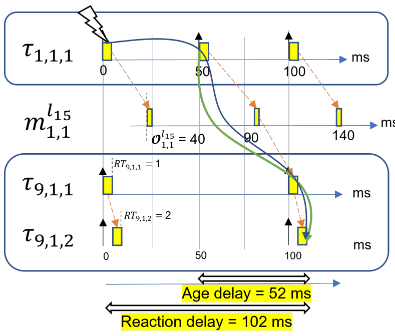
(a) $\mathcal{O}_{1,1}^{l_{15}} = 30\ ms$; $T_{9,1,1} = T_{9,1,2} = 50\ ms$.



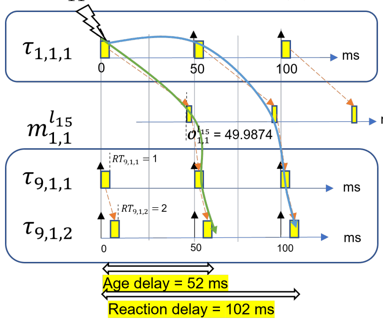
(b) $\mathcal{O}_{1,1}^{l_{15}} = 30\ ms$; $T_{9,1,1} = T_{9,1,2} = 100\ ms$.



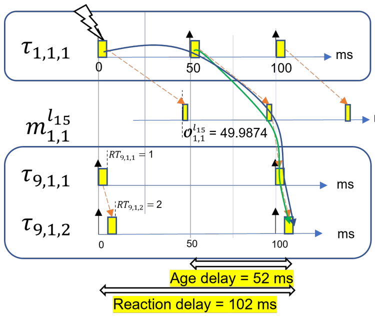
(c) $\mathcal{O}_{11}^{l_{15}} = 40\ ms$; $T_{9,1,1} = T_{9,1,2} = 50\ ms$.



(d) $\mathcal{O}_{11}^{l_{15}} = 40\ ms$; $T_{9,1,1} = T_{9,1,2} = 100\ ms$.



(e) $\mathcal{O}_{11}^{l_{15}} = 49.9874\ ms$;
$T_{9,1,1} = T_{9,1,2} = 50\ ms$.



(f) $\mathcal{O}_{11}^{l_{15}} = 49.9874\ ms$;
$T_{9,1,1} = T_{9,1,2} = 100\ ms$.

Figure 9.23. The effect of offset variations and harmonically set periods on the data age and reaction time delays of the transaction (offsets 30 ms, 40 ms and 49.987 ms).

Bibliography

[1] L. Lo Bello, R. Mariani, S. Mubeen, and S. Saponara. Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems. *IEEE Transactions on Industrial Informatics*, 2019.

[2] ISO 11898-1. Road Vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High-speed Communication, ISO Standard-11898, 1993.

[3] Christina Rödel, Susanne Stadler, Alexander Meschtscherjakov, and Manfred Tscheligi. Towards Autonomous Cars: The Effect of Autonomy Levels on Acceptance and User Experience. ACM, 2014.

[4] *802.1Q-2022 - IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks*. IEEE, 2022.

[5] 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.

[6] A. C. Rajeev, Swarup Mohalik, Manoj G. Dixit, Devesh B. Chokshi, and S. Ramesh. Schedulability and End-to-End Latency in Distributed ECU Networks: Formal Modeling and Precise Estimation. In *Proceedings of the Tenth ACM International Conference on Embedded Software*, EMSOFT '10, 2010.

[7] S. Mubeen, J. Mäki-Turja, and M. Sjödin. Support for End-to-End Response-Time and Delay Analysis in the Industrial Tool Suite: Issues, Experiences and a Case Study. *Computer Science and Information Systems*, 2013.

[8] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. End-to-end Timing Analysis of Cause-effect Chains in Automotive Embedded Systems. *Journal of Systems Architecture*, 2017.

[9] AUTOSAR Consortium, AUTOSAR Techincal Overview [online], Release 4.1, Rev.2, Ver.1.1.0., `http://autosar.org`.

[10] EAST-ADL Domain Model Specification, V2.1.12,. http://www.east-adl.-info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf.

[11] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System-level Performance Analysis - the SymTA/S Approach. *Computers and Digital Techniques*, 2005.

[12] S. Mubeen, H. Lawson, J. Lundbäck, M. Gålnander, and K. Lundbäck. Provisioning of Predictable Embedded Software in the Vehicle Industry: The Rubus Approach. In *4th International Workshop on Software Engineering Research and Industry Practice, located at the 39th International Conference on Software Engineering*. ACM, 2017.

[13] Robert Davis, Alan Burns, Reinder Bril, and Johan Lukkien. Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised. *Real-Time Systems*, 2007.

[14] S. Mubeen, J. Mäki-Turja, and M. Sjödin. MPS-CAN Analyzer: Integrated Implementation of Response-time Analyses for Controller Area Network. *Journal of Systems Architecture*, 2014.

[15] Mohammad Ashjaei, Gaetano Patti, Moris Behnam, Thomas Nolte, Giuliana Alderisi, and Lucia Lo Bello. Schedulability Analysis of Ethernet Audio Video Bridging Networks with Scheduled Traffic Support. *Real-Time Systems*, 2017.

[16] Saad Mubeen, Mohammad Ashjaei, and Mikael Sjödin. Holistic Modeling of Time Sensitive Networking in Component-based Vehicular Embedded Systems. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019.

[17] L. Lo Bello and W. Steiner. A Perspective on IEEE Time-sensitive Networking for Industrial Communication and Automation Systems. *Proceedings of the IEEE*, 2019.

[18] M. Ashjaei, L. Lo Bello, M. Daneshtalab, G. Patti, S. Saponara, and S. Mubeen. Time-sensitive Networking in Automotive Embedded Systems: State-of-the-Art and Research Opportunities. *Journal of Systems Architecture*, 2021.

[19] J. Diemer, D. Thiele, and R. Ernst. Formal Worst-case Timing Analysis of Ethernet Topologies with Strict-priority and AVB Switching. In *International Symposium on Industrial Embedded Systems*, 2012.

[20] U. D. Bordoloi, A. Aminifar, P. Eles, and Z. Peng. Schedulability Analysis of Ethernet AVB Switches. In *International Conference on Embedded and Real-Time Computing Systems and Applications*, 2014.

[21] X. Li and L. George. Deterministic Delay Analysis of AVB Switched Ethernet Networks Using an Extended Trajectory Approach. *Real-Time Systems*, 2017.

[22] J. Cao, P. J. L. Cuijpers, R. J. Bril, and J. J. Lukkien. Independent yet Tight WCRT Analysis for Individual Priority Classes in Ethernet AVB. In *International Conference on Real-Time Networks and Systems*, 2016.

[23] A. Finzi, A. Mifdaoui, F. Frances, and E. Lochin. Network Calculus-based Timing Analysis of AFDX Networks with Strict Priority and TSN/BLS Shapers. In *IEEE 13th International Symposium on Industrial Embedded Systems*, 2018.

[24] A. Finzi and A. Mifdaoui. Worst-case Timing Analysis of AFDX Networks with Multiple TSN/BLS Shapers. *IEEE Access*, 2020.

[25] D. Maxim and Y.-Q. Song. Delay Analysis of AVB traffic in Time-sensitive Networks (TSN). In *International Conference on Real-time Networks and Systems*, 2017.

[26] D. Thiele, R. Ernst, and J. Diemer. Formal Worst-case Timing Analysis of Ethernet TSN's Time-aware and Peristaltic Shaperss. In *Vehicular Networking Conference*, 2015.

[27] L. Zhao, P. Pop, Z. Zheng, and Q. Li. Timing Analysis of AVB Traffic in TSN Networks using Network Calculus. In *Real-Time and Embedded Technology and Applications Symposium*, 2018.

[28] L. Zhao, P. Pop, and S. Craciunas. Worst-case Latency Analysis for IEEE 802.1Qbv Time-sensitive Networks using Network Calculus. *IEEE Access*, 2018.

[29] M. Ashjaei, G. Patti, M. Behnam, T. Nolte, G. Alderisi, and L. Lo Bello. Schedulability Analysis of Ethernet Audio Video Bridging Networks with Scheduled Traffic Support. *Real-Time Systems*, 2017.

[30] G. Alderisi, G. Patti, and L. Lo Bello. Introducing Support for Scheduled Traffic over IEEE Audio Video Bridging Networks. In *IEEE International Conference on Emerging Technologies and Factory Automation*, 2013.

[31] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Afshar, and S. Mubeen. Schedulability Analysis of Best-effort Traffic in TSN Networks. In *26th IEEE International Conference on Emerging Technologies and Factory Automation*, 2021.

[32] D. Thiele and R. Ernst. Formal Worst-case Performance Analysis of Time-sensitive Ethernet with Frame Preemption. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation*, 2016.

[33] L. Zhao, P. Pop, Z. Zheng, H. Daigmorte, and M. Boyer. Latency Analysis of Multiple Classes of AVB Traffic in TSN with Standard Credit Behavior using Network Calculus. *IEEE Transactions on Industrial Electronics*, 2020.

[34] L. Lo Bello, M. Ashjaei, G. Patti, and M. Behnam. Schedulability Analysis of Time-Sensitive Networks with Scheduled Traffic and Preemption Support. *Journal of Parallel and Distributed Computing*, 2020.

[35] L. Zhao, F. He, E. Li, and J. Lu. Comparison of Time-sensitive Networking (TSN) and TTEthernet. In *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*. IEEE, 2018.

[36] M. Günzel, K. Chen, N. Ueter, G. von der Brüggen, M. Dürr, and J. Chen. Timing Analysis of Asynchronized Distributed Cause-Effect Chains. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2021.

[37] K. Gemlau, L. Köhler, R. Ernst, and S. Quinton. System-level Logical Execution Time: Augmenting the Logical Execution Time Paradigm for Distributed Real-Time Automotive Software. *ACM Transactions on Cyber-Physical Systems*, 2021.

[38] H. Teper, M. Günzel, N. Ueter, G. von der Brüggen, and J. Chen. End-to-end Timing Analysis in ROS2. In *2022 IEEE Real-Time Systems Symposium (RTSS)*, 2022.

[39] M. Ashjaei, S. Mubeen, J. Lundbäck, M. Gålnander, K. Lundbäck, and T. Nolte. Modeling and Timing Analysis of Vehicle Functions Distributed over Switched Ethernet. In *43rd Annual Conference of the IEEE Industrial Electronics Society*, 2017.

[40] S. Mubeen, M. Gålnander, J. Lundbäck, and K. Lundbäck. Extracting Timing Models from Component-based Multi-criticality Vehicular Embedded Systems. In *15th International Conference on Information Technology : New Generations*, 2018.

[41] M. Ashjaei, N. Khalilzad, S. Mubeen, M. Behnam, I. Sander, L. Almeida, and T. Nolte. Designing End-to-end Resource Reservations in Predictable Distributed Embedded Systems. *Real-time Systems*, 2017.

[42] J. Schlatow, M. Mostl, S. Tobuschat, T. Ishigooka, and R. Ernst. Data-age Analysis and Optimisation for Cause-Effect Chains in Automotive Control Systems. In *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, 2018.

[43] Jorge Martinez, Ignacio Sañudo, and Marko Bertogna. Analytical Characterization of End-to-end Communication Delays with Logical Execution Time. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[44] Tobias Klaus, Florian Franzmann, Matthias Becker, and Peter Ulbrich. Data propagation delay constraints in multi-rate systems: Deadlines vs. job-level dependencies. In *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, RTNS '18. ACM, 2018.

[45] R. Bi, X. Liu, J. Ren, P. Wang, H. Lv, and G. Tan. Efficient Maximum Data Age Analysis for Cause-effect Chains in Automotive Systems. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[46] Marco Dürr, Georg Von Der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. End-to-end Timing Analysis of Sporadic Cause-Effect Chains in Distributed Systems. *ACM Transactions on Embedded Computed Systems*, 2019.

[47] Tomasz Kloda, Antoine Bertout, and Yves Sorel. Latency Analysis for Data Chains of Real-time Periodic Tasks. In *2018 IEEE 23rd international conference on emerging technologies and factory automation (ETFA)*. IEEE, 2018.

[48] L. Köhler, P. Hertha, M. Beckert, A. Bendrick, and R. Ernst. Robust Cause-Effect Chains with Bounded Execution Time and System-Level Logical Execution Time. *ACM Transactions on Embedded Computing Systems*, 2022.

[49] S. Mubeen, J. Mäki-Turja, and M. Sjödin. Communications-oriented Development of Component-based Vehicular Distributed Real-time Embedded Systems. *Journal of Systems Architecture*, 2014.

[50] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, and K. Lundbäck. Supporting Timing Analysis of Vehicular Embedded Systems through the Refinement of Timing Constraints. *International Journal on Software and Systems Modeling*, 2017.

[51] S. Mubeen, M. Gålnander, A. Bucaioni, J. Lundbäck, and K. Lundbäck. Timing Verification of Component-based Vehicle Software with Rubus-ICE: End-user's Experience. In *2018 IEEE/ACM 1st International Workshop on Software Qualities and their Dependencies*. IEEE, 2018.

[52] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen. Synthesising Schedules to Improve QoS of Best-effort Traffic in TSN Networks. In *29th International Conference on Real-time Networks and Systems*, 2021.

[53] H. Lim, K. Weckemann, and D. Herrscher. Performance Study of an In-car Switched Ethernet Network without Prioritization. In *Communication Technologies for Vehicles*, 2011.

[54] S. Kramer, D. Ziegenbein, and A. Hamann. Real World Automotive Benchmarks for Free. In *6th Intl. Workshop on Analysis Tools and Methodologies for Embedded and Real-Time Systems*, 2015.

Chapter 10

Paper E: End-to-end Timing Modeling and Analysis of TSN in Component-Based Vehicular Software

Bahar Houtan, Mehmet Onur Aybek, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, John Lundbäck, Saad Mubeen.
In the 25th IEEE International Symposium on Real-Time Distributed Computing, 2023.

Abstract

In this paper, we present an end-to-end timing model to capture timing information from software architectures of distributed embedded systems that use network communication based on the Time-Sensitive Networking (TSN) standards. Such a model is required as an input to perform end-to-end timing analysis of these systems. Furthermore, we present a methodology that aims at automated extraction of instances of the end-to-end timing model from component-based software architectures of the systems and the TSN network configurations. As a proof of concept, we implement the proposed end-to-end timing model and the extraction methodology in the Rubus Component Model (RCM) and its tool chain Rubus-ICE that are used in the vehicle industry. We demonstrate the usability of the proposed model and methodology by modeling a vehicular industrial use case and performing its timing analysis.

10.1 Introduction

Designing highly software-intensive vehicular embedded systems is challenging due to the enormous size and complexity of the software [1]. In addition, the complexity in modern vehicular software is continuously growing by integration of network communication mechanisms introduced by Time-Sensitive Networking (TSN) standards [2, 3, 4]. TSN standards are attractive solutions to address the high-bandwidth and real-time communication requirements in vehicular applications. These standards enhance switched Ethernet with deterministic traffic shaping mechanisms. More specifically, TSN supports hard real-time, high bandwidth with low-latency and low-jitter traffic transmission. These features are supported by offline scheduled time-triggered traffic enabled by the Time-Aware Shaper (TAS), resource reservation for different classes of traffic by a Credit-Based Shaper (CBS), and clock synchronization [5]. Although TSN standards provide flexible design of complex and deterministic network communication, this flexibility comes with the cost of further complicating the design process due to including several configurable factors for network devices and traffic.

Model-based Engineering (MBE) and Component-based Software Engineering (CBSE) approaches [6, 7] are widely being used to manage the software complexity and for cost-effective development of vehicular software systems. There are several domain-specific languages and component models that can be used to model software architectures of vehicular systems, e.g., AUTomotive Open System ARchitecture (AUTOSAR) [8], Rubus Component Model (RCM) [9], AMALTHEA[1][10], EAST-ADL [11], and Architecture Analysis & Design Language (AADL) [12], to name a few. All the models and languages assist the embedded software developers by structuring necessary information to develop and design complex distributed embedded software systems.

In addition to managing the design complexity of vehicular distributed software systems, the developers of these systems are required to verify real-time requirements that are specified on their software architectures. The timing requirements can be verified by performing the end-to-end data-propagation delay analysis of the software architectures of these systems [13, 14]. The analysis results can also guide in the refinement of the software architectures [15]. This analysis is already implemented in several tools in the vehicle industry that support model- and component-based development of these systems, e.g., SymTA/S[2][16] and Rubus [17]. The analysis requires the end-to-end timing model as a crucial input. The end-to-end timing model includes comprehensive timing information of the vehicular distributed software system. The instances of this timing model must be extracted from the software architectures of

[1]https://itea4.org/project/amalthea.html

[2]SymTA/S tool has been acquired by Luxoft (`https://www.luxoft.com`)

these systems and provided as an input to the model-based analysis tools.

In this paper, we present an end-to-end timing model to represent a distributed embedded system with TSN support. Moreover, we present an automated methodology to systematically extract instances of the end-to-end timing model from the software architectures of these systems. The end-to-end timing model can be used for end-to-end data-propagation delay analysis of the systems. We evaluate the proposed methodology on an industrial use-case from the vehicular domain based on RCM developed in Rubus-ICE tool suite [18]. As a proof of concept, we chose RCM because it is the first commercially available model that supports comprehensive modeling and end-to-end data-propagation delay analysis of TSN-based vehicular distributed embedded systems [19].

The **main contributions** in this paper are as follows.

- We propose an end-to-end timing model to describe TSN networks with all configuration parameters in the distributed vehicular embedded software systems.
- We provide an automated methodology to extract instances of the end-to-end timing model from component-based software architectures of vehicular systems.
- We provide a proof of concept for the proposed timing model and extraction methodology by integrating them with the component-based software engineering environment of an industrial tool suite, namely Rubus-ICE.
- We evaluate the proposed model and methodology as well as their integration with the Rubus-ICE tool suite on a vehicular industrial use-case.

10.2 Background and Related Works

As described in [5], model-based software development of vehicular embedded systems generically consists of four steps: (1) modeling the software architecture; (2) extracting timing properties and requirements from the software architecture; (3) validate the timing requirements and constraints using end-to-end data-propagation delay analysis; and (4) refining the software architecture according to the timing analysis results.

10.2.1 Modeling the Software Architecture

There are several software architecture modeling languages and component models, such as AUTOSAR, RCM, AMALTHEA, EAST-ADL, AADL, to mention a few.

AUTOSAR is widely used for developing software architecture for automotive systems. SymTA/S is a commercial timing analysis and optimization framework that complies with AUTOSAR. A recent work in [20] integrates AUTOSAR adaptive with applicable standards to develop more sophisticated systems. The proposed three-layer architecture coordinates a binding between AUTOSAR Adaptive, OPC UA standards[3], and TSN standards. The paper argues that the proposed architecture lacks maturity since all the involved technologies in the architecture layers are still under development.

RCM comprises of hierarchical entities which are necessary to model a distributed embedded system that supports TSN. At the highest level, the system contains at least two nodes and a network element that interconnects the nodes. A node (end-station) is a processing element that provides run-time environment for one or more Software Applications (SA). The SA provides spacial and temporal isolation to the part of overall software architecture within the system. In RCM, a software architecture is modeled by interconnecting a set of Software Components (SWCs). An SWC is a design-time entity that corresponds to a task at run-time or in the timing model. The SWCs communicate with each other by their interfaces (a set of data and trigger ports).

To the best of our knowledge, RCM is the first and only component model that supports comprehensive modeling of TSN [19]. Recently, there have been some efforts in increasing the performance and applicability of the other modeling approaches to RCM by model transformation [21]. The work in [22] proposes a mapping technique between AMALTHEA and RCM to enable timing analysis of AMALTHEA-based component models in RCM. In addition, the work in [23] presents a mapping from EAST-ADL models to RCM with the aim of enabling the timing analysis of a non-RCM model.

10.2.2 Timing Information Extraction from Software Architectures

Modeling and extraction of timing models from the software architectures of component-based vehicular embedded systems that are developed with RCM are presented in [24]. The work in [24] considers several onboard communication protocols like Controller Area Network (CAN) [25], CANopen [26], AUTOSAR COMM [27] and switched Ethernet. However, it does not consider TSN, which is the main focus of our work.

The work in [19] complements [24] by integrating several aspects of modeling TSN standards in a component model with timing analysis perspective. However, the presented TSN timing model in [19] and the timing model extraction in [24] are not expressive enough to include all the timing aspects of TSN. Nevertheless, the complexity of TSN requires significant automation and optimization in the timing information extraction and configuration process prior to the end-to-end data-propagation delay analysis.

[3]https://opcfoundation.org/about/opc-technologies/opc-ua/

A recent work [28] discusses preliminary ideas and work-in-progress on extraction of timing models from TSN-based software architectures. In comparison, we propose a comprehensive end-to-end timing model and an automated methodology for the extraction of end-to-end timing models from the software architectures of vehicular distributed embedded systems, which is complemented by an automotive application case study. Moreover, our work takes into account the integration and configuration aspects of the TSN timing models.

10.2.3 End-to-end Timing Models, Timing Analysis and Verification

In order to verify the timing requirements of the chains of tasks (SWC at the design time) and network messages in the distributed embedded systems, end-to-end data-propagation delays (data age and reaction time delays) are calculated and compared with the corresponding data age and reaction time constraints. The timing constraints corresponding to the data age and reaction time delays are part of the timing model of AUTOSAR standard. The data age delay is the time elapsed between the arrival of data at the input of the chain and the latest availability of the corresponding data at the output. Whereas, the reaction time delay corresponds to the earliest availability of the data at the output of the chain corresponding to the data that just missed the read access (of the event) at the input of the chain [29, 13].

The data age and reaction time delays in a two-node distributed embedded system which is modeled based on the conventional task model and with one TSN message are demonstrated in Figure 10.1. The data flows between the task instances and messages are indicated by dashed orange arrows in Figure 10.1 to indicate the read and write of data in the chain.

There are several works that have also presented end-to-end data-propagation delay analysis based on the conventional task model [14, 30]. The work in [17, 15] considered the analysis for distributed embedded systems that are based on CAN network. The work in [31] considers the systems that use Ethernet networks. The works in [14] and [32] presented the end-to-end data-propagation delay analysis for vehicular applications, where some of the techniques have been implemented in tools to support component-based software development, e.g., [17] and [33]. The works in [34, 18] present open research challenges and their solutions when integrating the timing analysis with model-based development tools. However, these works only focus on traditional onboard networks such as CAN without considering TSN.

In the case of TSN, there are several additional features that need to be considered such as synchronization of the end-stations, and the presence of different shaping mechanisms for different TSN classes. A recent work in [35] shows that the existing end-to-end data-propagation delay analysis supports the non-scheduled TSN classes, but it does not support the scheduled traffic in TSN.

The end-to-end data-propagation delay analysis results are typically presented in the form of tables showing individual response time of each task (SWC) or message and the data age and reaction time delays. Such information can be used by the system designer and integrator to refine the software architecture [15]. Automated refinement of the software architecture according to the analysis results is an open research challenge [5] that could be beneficial for improving the scalability and efficiency of the model-based software development for more sophisticated vehicular embedded systems.

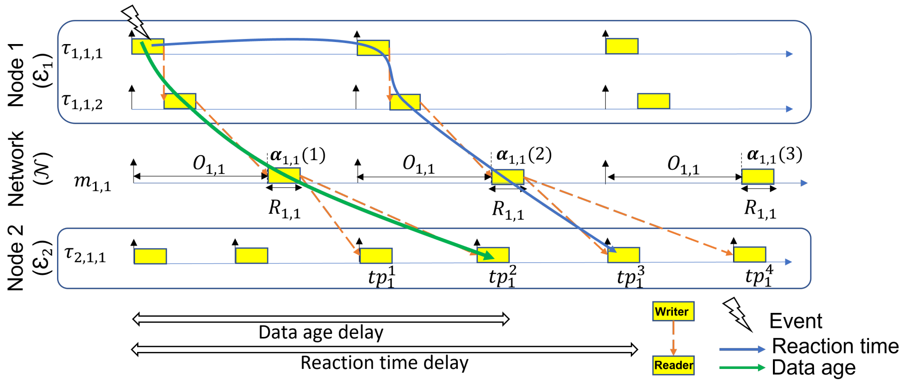


Figure 10.1. Demonstration of data age and reaction time delays.

10.3 End-to-end Timing Model

In this paper, we consider the end-to-end timing model that captures the timing information from the software architecture of TSN in the distributed embedded systems. Such a model is a necessary input for the analysis techniques and tools to verify the timing behavior of the software architecture. The system model consists of two or more end-stations (or nodes), denoted by $\mathcal{E}$, and at least one TSN network, denoted by $\mathcal{N}$ as shown in Eq. (10.1):

$$\mathcal{S} := \langle \mathcal{E}, \mathcal{N} \rangle \tag{10.1}$$

The set of networks in the system model is defined by Eq. (10.2) as follows:

$$\mathcal{N} := \{\mathcal{N}_1, ..., \mathcal{N}_{|\mathcal{N}|}\} \tag{10.2}$$

The networks connect end-stations to each other. The overall set of end-stations in the system model is denoted by $\mathcal{E}$, as shown in Eq. (10.3):

$$\mathcal{E} := \{\mathcal{E}_1, ..., \mathcal{E}_{|\mathcal{E}|}\} \tag{10.3}$$

Various components of the end-to-end timing are discussed in the following subsections.

10.3.1 End-station Timing Model

The system model considers single-core end-stations similar to the model in [19]. In such a system, the computation and storage resources need to be shared between the SWCs in an isolated manner. A set of SWCs contributing to a software functionality are isolated in an application within the end-station.

The end-station consists of one or more applications as shown in Eq. (10.4). The application provides spacial and temporal isolation of the software functionality. In other words, the applications are isolated in terms of allocated time for execution, and also allocated memory.

$$\mathcal{E}_i := \{A_{i1}, ..., A_{i|\mathcal{E}_i|}\} \tag{10.4}$$

Above, i indicates the end-station ID that the application belongs to. $|E_i|$ is the total number of applications in the end-station. A_{ij} specifies the application j of the end-station i. Besides, the criticality level of each application is denoted by C_{ij}. Each application includes one or more tasks as shown in Eq. (10.5).

$$A_{ij} := \left\langle \{\tau_{ij1}, ..., \tau_{ij|A_{ij}|}\}, C_{ij} \right\rangle \tag{10.5}$$

A task, denoted by τ_{ijk}, has the attributes as shown in Eq. (10.6):

$$\tau_{ijk} := \langle C_{ijk}, T_{ijk}, O_{ijk}, P_{ijk}, J_{ijk}, B_{ijk}, R_{ijk}, D_{ijk} \rangle \tag{10.6}$$

where k is the ID of the task. The attributes of a task are specified as follows. C_{ijk} presents the worst-case execution time of the task, T_{ijk} denotes the period, and O_{ijk} represents the offset. Moreover, P_{ijk} is the priority, J_{ijk} is the release jitter, and B_{ijk} is the blocking time experienced by the task. Finally, R_{ijk} denotes the worst-case response time and D_{ijk} presents the deadline of the task. Note that we consider a one-to-one mapping between a SWC (a design-time entity) in the component model and a task (run-time entity) in an end-station in the end-to-end timing model.

10.3.2 TSN Network Timing Model

The properties of the network $\mathcal{N}_i$ are as follows.

$$\mathcal{N}_i := \langle s_i, \mathcal{L}_i, \mathcal{T}_i, \mathcal{M}_i, Slope, Preemption \rangle \tag{10.7}$$

where, s_i is the network speed. $\mathcal{L}_i$ is the set of links connecting the end-stations to TSN switches and the TSN switches to each other. $\mathcal{T}_i$ indicates TSN classes through which the messages are communicated. The set $\mathcal{M}_i$ holds the TSN network's messages. The available TSN classes are shown in Eq. (10.8).

$$\mathcal{T}_i = \{ST, AVB, BE\} \tag{10.8}$$

where, ST is the highest priority traffic (Scheduled Traffic) which is scheduled offline with offsets. The class AVB (Audio-Video Bridging) associates multiple priorities (up to maximum 8 priorities) to the TSN port's queues which undergo the CBS mechanism. In AVB class, priorities are specified in alphabetical order and can be assigned to all the egress queues. For example, class A has a higher priority than class B. Class BE is the lowest priority class (Best-effort) which often does not have any timing requirements. Each queue in TSN port can be set to operate under only one of these classes.

A message belonging to the set of messages $\mathcal{M}_i$ is denoted by m_{ij}, where i shows the ID of the network the message belongs to and j is the ID of the message within that network. The definition of the message in the end-to-end timing model maps to the same entity in the component model.

$$m_{ij} := \langle \mathcal{X}_{ij}, C_{ij}, P_{ij}, Size_{ij}, T_{ij}, \mathcal{L}_{ij}, J_{ij}, B_{ij}, R_{ij}, \mathcal{O}_{ij}, D_{ij} \rangle \tag{10.9}$$

where, $\mathcal{X}_{ij}$ is the transmission mode which can be periodic or sporadic. C_{ij} is the worst-case transmission time, P_{ij} is the priority, and $Size_{ij}$ is the payload size. The period is shown by T_{ij}. In the case of sporadic transmission, T_{ij} represents the minimum inter-arrival time between any two instances of the message. The set of links in the route of the message is shown by $\mathcal{L}_{ij}$. The release jitter is shown by J_{ij} and B_{ij} denotes the blocking time which contributes to calculating the response time shown by R_{ij}. Moreover, $\mathcal{O}_{ij}$ is the offset of the message in case the priority of the message is ST, and finally, D_{ij} represents the deadline of the message.

The set of slopes denoted by $Slope$ holds the set of idle slopes for each AVB port queue connected to a link in the network ($\mathcal{L}_i$). $Slope$ values are used to allocate bandwidth for each queue assigned to AVB class. Moreover, the $Preemption$ set holds the set of flags per link in the network ($\mathcal{L}_i$). This list specifies whether the corresponding link to each member of the set allows pausing transmission of an ongoing frame in favor of a higher priority frame.

10.3.3 Transactional Model

For the sake of end-to-end analysis, the timing model follows the transactional model in [36] to link the instances of the tasks and messages into one transaction. A set of transactions is indicated by Γ, and a transaction, denoted by Γ^d, contains a chain of tasks and messages that allow flow of data within the system. The superscript d indicates the transaction ID. The chain can either contain several tasks that are within one end-station or a set of tasks distributed over multiple end-stations that communicate through the network. An end-station may include one or more of its tasks in the transaction. The attributes of a transaction are shown in Eq. (10.10).

$$\Gamma^d := \left\langle Chain^d, T^d, a^d, r^d, Cr^d \right\rangle \tag{10.10}$$

where the period of the transaction is T^d. The notations r^d and a^d are the transaction's reaction time and data age delays that are calculated by the end-to-end data-propagation delay analysis. Besides, Cr^d specifies the set of timing constraints on the reaction time and data age delays of the transaction. $Chain^d$ represents the chain of tasks in transaction Γ^d as shown in Eq. (10.11):

$$Chain^d := (\{\tau^d_{ij\alpha}, \ldots, \tau^d_{ij\Omega}\}, m^d_{i1}, \{\tau^d_{nj\alpha}, \ldots, \tau^d_{nj\Omega}\}, \ldots, m^d_{ix}, \{\tau^d_{kj\alpha}, \ldots, \tau^d_{kj\Omega}\}) \tag{10.11}$$

In Eq. (10.11), the order of tasks from left to right shows the direction of the data flow, which starts from the initiator task ($\tau^d_{ij\alpha}$), and ends in the terminator task ($\tau^d_{kj\Omega}$) of the transaction.

A transaction can comprise of a set of tasks of one end-station. In such a case, this end-station is assumed to be initiator and terminator of the transaction. Moreover, a transaction may comprise of tasks that are located on different end-stations, then two adjoining tasks in the transaction belonging to two different end-stations communicate via network messages. All the tasks in the transaction that belong to the transaction's initiator end-station ($\mathcal{E}_i$) are specified at the beginning of the transaction. Similarly, all the tasks belonging to the transaction's terminator end-station ($\mathcal{E}_k$) are specified at the end of the transaction. There can be one or more intermediary end-stations, such as ($\mathcal{E}_n$) in Eq. (10.11), that are part of the transaction. The first message in transaction is sent from the initiator end-station. The message sent from each end-station is placed at the right hand side of the set of end-station's tasks in the transaction, immediately after the source task.

In a TSN network, a message can pass through a set of links, which are specified by the set ($\mathcal{L}_{ij}$). Since, multiple end-stations can be engaged in a transaction, multiple messages must be communicated in a transaction. We assume that there can be a set of x number of messages in the transaction. The terminator end-station receives the last message, by the left most task in its set of tasks. Since, tasks and messages can be used by multiple transactions the superscript d shows the transaction ID that uses the task or message. In Eq. (10.11), m^d_{ix} denotes the message x that is activated by end-station i, and is involved in the data flow represented by chain d.

10.3.4 Timing Requirements Model

The timing constraint on the transaction, Cr^d, defines the maximum allowed value of the delays, such as data age (Age^d), and reaction time ($Reac^d$). These constraints are shown in Eq. (10.12) for the transaction Γ^d:

$$Cr^d := \{Age^d, Reac^d\} \tag{10.12}$$

10.4 Proposed End-to-end Timing Model Extraction Methodology

The proposed methodology aims at automated extraction of end-to-end timing information from component-based software architectures of TSN-based distributed embedded systems. The extracted timing information is populated in an instance of the end-to-end timing model, which is fed as input to the end-to-end data-propagation analysis engines. An end-user, also known as the software architecture developer/modeler, is able to interact with the component model directly by manually modeling the software architecture. The software architecture can be developed using any component model. As a proof of concept, we use RCM to model the software architecture. The end-user often has limited or no knowledge about detailed timing information in the system, but is skilled in designing software architectures using component models. In order to retrieve the required timing information to analyze and verify the timing requirements on the software architectures, we propose a timing model extraction methodology that is conceptualized in Figure 10.2.

The *end-user* shown in module *a* in Figure 10.2 develops the software architecture with the help of the component models as shown in module *b* in Figure 10.2. Accordingly, the *end-to-end timing model extraction* shown in module *c* in Figure 10.2, coordinates the extraction of an instance of the end-to-end timing model from the software architecture. Some information can be implicitly obtained from the software architecture, hence there is no need for detailed specification of such information by the software architecture developer. Furthermore, some information includes variable parameters that can be refined in the software architecture through the *configuration* step, as shown by module *b2* in Figure 10.2. The configuration step is performed while developing the software architecture. Besides, this step could also be iterated by the system's integrator/configurator or by the automated configuration tools.

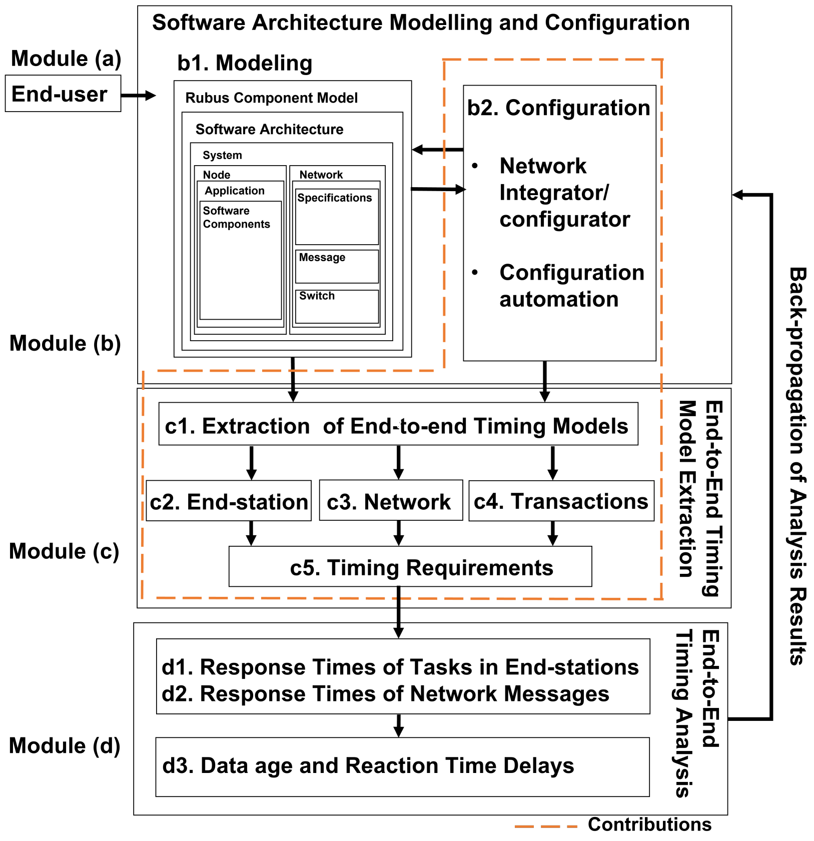

Software Architecture Modelling and Configuration
Module (a)
End-user
b1. Modeling
Rubus Component Model
Software Architecture
System
Node
Application
Software Components
Network
Specifications
Message
Switch
b2. Configuration
• Network Integrator/ configurator
• Configuration automation
Module (b)
c1. Extraction of End-to-end Timing Models
c2. End-station
c3. Network
c4. Transactions
c5. Timing Requirements
Module (c)
End-to-End Timing Model Extraction
d1. Response Times of Tasks in End-stations
d2. Response Times of Network Messages
d3. Data age and Reaction Time Delays
Module (d)
End-to-End Timing Analysis
Back-propagation of Analysis Results
Contributions

Figure 10.2. Timing model extraction methodology.

The module *c1* extracts the end-to-end timing information from the model of distributed software architecture that is augmented with configuration information. The sources of end-to-end timing information can be classified as follows:

1. **User-defined (User):** The properties in this category are extracted from the input specified by the end-user while developing the software architecture. The end-user may not have knowledge about the detailed timing aspects of the system.

2. **Software-architecture-derived (SWA-d):** This timing information includes the properties that are either inherited from other components in the software architecture, calculated according to the user-defined properties or implicitly initialized based on other properties set by other sources.

3. **Configurable (Conf.):** This category holds the properties obtained from the software architecture, which are configured according to some logical constraints, and algorithms for the sake of optimizing timing performance of the system. Such parameters could also be defined by system experts based on their knowledge of the system requirements, i.e., system configurators or integrators.

4. **Analysis-derived (Analysis):** The values of the properties in this category are obtained by performing various analyses, e.g., response-time analysis of individual end-stations, response-time analysis of TSN network, and end-to-end data-propagation analysis.

In the end-to-end timing model extraction step, module *c1* in Figure 10.2, the properties obtained from various sources in module *b*, i.e., the software architecture developer, software architecture and configurators, are mapped to the parameters in the proposed end-to-end timing model as shown in Table 10.1. In the next subsections, we explain the end-to-end timing model specification for each entity, as shown in modules *c2*, *c3*, *c4* and *c5* of Figure 10.2.

Finally, the extracted end-to-end timing model is fed into the *end-to-end timing analysis* as shown in module *d* in Figure 10.2. Furthermore, under the procedures in module *d1* and *d2* in Figure 10.2 the response times of the tasks and messages, and subsequently the end-to-end delays, i.e., data age and reaction time delays are calculated under module *d3* in Figure 10.2. Subsequently, the results of the end-to-end timing analysis are propagated back to the distributed software architecture model.

10.4.1 Specification of End-station Timing Information

In this step, each extracted property from the node component is specified to the associating entity, namely end-station in the proposed end-to-end timing model.

The software architecture developer (end-user) designs the software architecture's hierarchy, which includes the connections between the end-stations and the arrangement of the SWCs within an end-station. Therefore, an application (A_{ij}) and assignment of the tasks to each application is done by the software architecture developer. The criticality level (C_{ij}) of each application is associated to the priority of the tasks included in the application therefore the criticality level of the application is derived from the software architecture according to the priority of the tasks included in this application.

The parameters of the end-station that can be set directly by the end-user include the worst-case execution time (C_{ijk}) and the task priority (P_{ijk}). These user-defined parameters can be further used for obtaining the rest of the parameters in the classes software-architecture-derived, configurable and analysis-derived.

In case of considering implicit deadlines in the system, the deadline of the task (D_{ijk}) is equal to the task's period. As a result, deadline value is derived from the software architecture. The deadline can also be assigned according to the system timing information after performing the end-to-end timing analysis engines. Therefore, the deadline parameter can be also considered as configurable parameter.

Task's period (T_{ijk}) assignment is dependent on whether the task is independently or dependently triggered. If the task is independently triggered, then the task's period is user-defined. If the task is dependently triggered, the period of the task is inherited from its predecessor entity (task or message), which triggers the task.

Task's offset (O_{ijk}) is defined according to the pseudo code in Algorithm 1. The offset of a task is a configurable property in case the task is sending/receiving a message through the ST class in TSN. If the task is not connected to the network, task offset is derived from the software architecture. Task offset influences the quality of service of the system. The offsets can be defined in a manner to reduce the end-to-end delays (age and reaction) of the transactions. Besides, there are various other algorithms in literature which consider different optimization criteria for setting offsets of the tasks in distributed systems [37].

Finally, the release jitter (J_{ijk}), blocking (B_{ijk}) and response time (R_{ijk}) of the task are retrieved from timing analysis techniques and corresponding engines, such as [38].

10.4.2 Specification of Network Timing Information

In case of the network, the network speed (s_i), set of links ($\mathcal{L}_i$) and the available TSN classes ($\mathcal{T}_i$) are properties that are dependent on the design of the network within the software architecture.

The set of idle slopes (e.g. $slope_A$ and $slope_B$) can be chosen simply by globally assigning a value that applies to all the links in the TSN network, or it can be individually assigned for all the links that use AVB traffic, e.g., according to the load of AVB queues on each of the links. Consequently, the configuration of the idle slopes requires to deal with a trade-off between the simplicity of configuring the network versus the performance achieved by the configuration of the network.

Finally, the $Preemption$ is the last configurable parameter set which can be enabled/disabled to optimize the utilization of the links by the lower priority messages [39]. The pseudo code in Algorithm 2 shows the strategy to define the idle slopes and the

Algorithm 1 Specifying task offset.

```
for each $\Gamma^d \in \Gamma$ do
    for each $\tau^d_{ijk} \in \Gamma^d$ do
        if $\tau^d_{ijk} \in$ initiator $\mathcal{E}_i$ then
            if $\tau^d_{ijk}.O_{ijk}$ is configurable then
                $\tau^d_{ijk}.O_{ijk}$ is manually set or automatically optimized.
                $ST$ class is used.
            end
            else if $\tau^d_{ijk}.O_{ijk}$ is SWA-d then
                No offset is needed for the task.
                Release the task at time 0.
                A non-$ST$ class is used.
            end
        end
        else if $\tau^d_{ijk} \in$ terminator $\mathcal{E}_i$ then
            if $\tau^d_{ijk}$ receives $ST$ messages then
                $\tau^d_{ijk}.O_{ijk}$ manually set or automatically optimized according to the
                received message's offset at its last link.
            end
            else if $\tau^d_{ijk}$ receives non-$ST$ messages then
                No offset is needed for the task. Release the task at time 0.
            end
        end
    end
end
```

$Preemption$ set on the TSN links.

We assume that TSN messages inherit properties such as transmission mode ($\mathcal{X}_{ij}$), period (T_{ij}) (for Event triggered chains) and priority (P_{ij}) from the message's sender task in the transmitter end-station. The deadline of the message (D_{ij}) is assigned implicitly equal to its period. The deadline of the messages can be configured according to the same assumption for the tasks' deadline. The payload size ($Size_{ij}$) of the message is derived from the software architecture since it is calculated based on the payload size and the network speed. The worst-case transmission time (C_{ij}) of the message is derived from the software architecture based on payload size ($Size_{ij}$) of the message, and network speed (s_i).

The set of links that the message passes from the transmitter end-station to reach the receiver end-station, denoted by $\mathcal{L}_{ij}$, are derived from the software architecture. In some cases, there is only one route that delivers the message to the destination end-station, therefore it is possible for the software architecture developer to specify the routes manually. However, in many cases, there are more than one route from the transmitter end-station to the destination end-station. Consequently, the set of links that the message traverses can be configured to optimize the routing of the message. For example, some works define routes simply based on the shortest path [40]. On the other hand, TSN messages can also be re-routed with respect to the potential failure of the links [41] or according to the load of each traffic class in the network, i.e, ST or AVB [42]. Hence, the set of routes ($\mathcal{L}_{ij}$) is also specified as a configurable parameter.

As the pseudo code in Algorithm 3 indicates, the message offsets per link, denoted by the set $\mathcal{O}_{ij}$, are defined according to the priority of the message. If the message is assigned to class ST, there is a set of offsets per links between the source to the destination of the message. This set can be configured based on different optimization approaches. For instance, the optimization algorithm presented in [40] schedules the ST traffic in a way to reduce the end-to-end delay in the non-ST traffic. The optimization approaches for scheduling the ST traffic can be found in a recent survey [43]. If the message is not ST, the set of offsets per link ($\mathcal{O}_{ij}$) will not be used and contains all zeroes.

Algorithm 2 Specifying slope and preemption.

```
if simple link setup is desired then
    Set the idle slopes and preemption GLOBALLY.
end
else if optimized link setup is desired then
    Set the idle slopes and preemption PER LINK.
end
```

10.4.3 Specification of Transaction Timing Information

The transactions are specified by the software architecture developer in order to analyze a chain of tasks and messages ($Chain^d$) in the system. Tasks in a transaction can be triggered in two modes, i.e., "independent" or "dependent". Independently triggered tasks are activated based on their individual clocks or trigger sources, whereas dependently triggered tasks are activated by their predecessor tasks. The trigger mode for the messages are different depending on the TSN class defined for the message. If the message is ST, it is triggered independently based on static offsets defined for it per link specified in its route to the destination end-station. In case of the dependent trigger mode, the message is triggered right after the execution of its predecessor task is finished. Messages assigned to non-ST classes, such as AVB or BE, are activated dependently.

The transaction period (T^d) is calculated by finding the Least Common Multiple (LCM) of the periods of all tasks within the transaction.

10.4.4 Specification of Timing Requirements

All the timing constraints in the transaction, denoted by Cr^d, and end-to-end deadline can be either specified by the software architecture developer when defining the transaction; or they can be configured according to the network/system analysis. The reaction time (r^d) and data age (a^d) delays of the transaction d are calculated by the end-to-end data-propagation delay analysis.

Algorithm 3 Specifying message offsets.

```
if $m_{ij}.P_{ij}$ is $ST$ then
    Message offset ($m_{ij}.\mathcal{O}_{ij}$) is configurable.
    Optimize message offsets per link.
end
else if $m_{ij}.P_{ij}$ is not $ST$ then
    SWA-d mode.
    Message has no offset.
end
```

10.5 Evaluation on a Vehicular Industrial Use-case

The end-to-end timing model extraction methodology presented in the previous section is implemented as a proof of concept in the Rubus-ICE tool suite. In this section, we take advantage of a real vehicular use-case to validate the applicability of the proposed end-to-end timing model extraction methodology. For the sake of evaluations, we first model a software architecture of a distributed embedded system, consisting of a set of transactions, in RCM. Then we extract the end-to-end timing model from the software architecture in Rubus-ICE. Using the extracted timing model, we perform the end-to-end data-propagation delay analysis of the software architecture using the implemented analysis in Rubus-ICE [35] and discuss the analysis results.

10.5.1 Use-case Setup

The network topology in the use-case consists of nine end-station that are interconnected via three TSN switches as depicted in Figure 10.3. In the topology, all the end-stations are capable of generating TSN traffic, whereas HU and AVSink only receive TSN traffic from the other end-stations. The set of transactions and network traffic in this use-case are presented in Table 10.2. In summary, there are a set of 10 distributed chains (transactions). Each transaction includes tasks from two different end-stations and one message between the end-stations. The initiator end-station has only one task that sends the message to the network. The transaction terminator end-station includes maximum two tasks. On the receiver's side, the first task receives the TSN message and activates the second task in the receiver end-station. We assume that all tasks in an end-station belong to the same application.

The transactions 1 to 4 are using ST class and the transactions 5 to 10 are using BE class in the network. The offset of the sender tasks in transactions 1 to 4 are set to the default value which is 0. Accordingly, the messages transmitted from these tasks are assigned to ST class. Finally, the idle slopes are set to 0 for all the links since the AVB classes are not used in the use-case. The reaction time (Reac_d) and data age (Age_d) constraints on all the transactions are subsequently 70 ms and 45 ms. These constraints are specified by expert integrators from the industry (providers of the use-case).

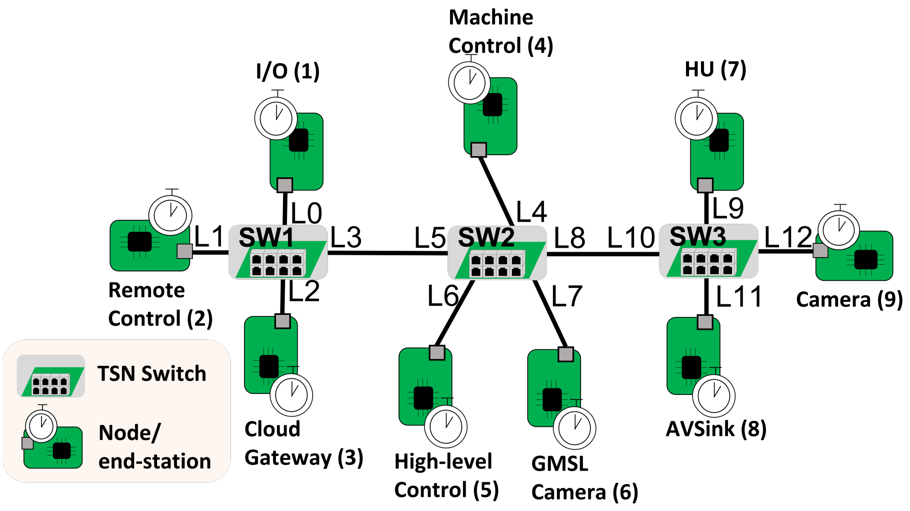


Figure 10.3. Use-case from the vehicular domain.

10.5.2 Modeled Use-case in Rubus-ICE

The system-level software architecture of the use-case modeled with RCM is depicted in Figure 10.4. The system-level software architecture consists of nine node models that correspond to the nine end-stations. All the nodes are connected to one TSN network model, as shown in Figure 10.4. In the internal model of the TSN network as shown in Figure 10.5, the flow from all the sender nodes are either towards HU or AVSink (the only sink nodes within the system).

The internal software architecture of each node is depicted in Figure 10.6, where the RCM representation of the set of SWCs within each node is shown by yellow components. For example, the software architecture of the Camera node consists of two SWC, where SWC1 in the Camera node that is used in transaction 3 (represented by $\tau^{3}_{9,1,1}$ in the end-to-end timing model) has period of 10 ms and sends an ST message with the priority of 2. Besides, SWC2 in the Camera node is used in transaction 5 (represented by $\tau^{5}_{9,1,2}$ in the end-to-end timing model) and has a lower priority than SWC1 (priority 1). The period of SWC2 is 20 ms. Transaction 3 is initiated from the SWC1 ($\tau^{3}_{9,1,1}$) in the Camera node and is terminated at the SWC2 in the HU node ($\tau^{3}_{7,1,2}$), which has the priority of 4. The terminator task of the transaction ($\tau^{3}_{7,1,2}$) is triggered by the SWC1 in the HU node ($\tau^{3}_{7,1,1}$). As a result, the terminator task inherits the period of its predecessor task, namely SWC1 in the HU node ($\tau^{3}_{7,1,1}$).

10.5.3 End-to-end Response-Time Analysis Results

Table 10.3 presents the results of the end-to-end data-propagation delay analysis. The results include reaction time and data age delays of the transactions as well as response times of the network messages. For instance, transaction 3 contains an ST message with the response time of 0.025 ms. The data age and reaction time delays of transaction 3 are subsequently 20.100 and 30.100 ms. Transaction 5 is also initiated at the Camera node, though from a different SWC, namely (SWC2), and it is terminated at SWC4 in HU. Transaction 5 uses the class BE of the TSN network. The calculated response time of the message in Transaction 5 is 0.153 ms. Besides, the data age and reaction time delays of transaction 5 are subsequently 40.300 and 60.300 ms. As the specified data age and reaction time constraints on all transactions are 70 ms and 45 ms respectively, it is evident from Table 10.3 that all transactions meet their specified timing constraints.

10.6 Conclusions

In this paper, we presented a detailed end-to-end timing model of vehicular distributed embedded systems that utilize Time-Sensitive Networking (TSN) standards. This

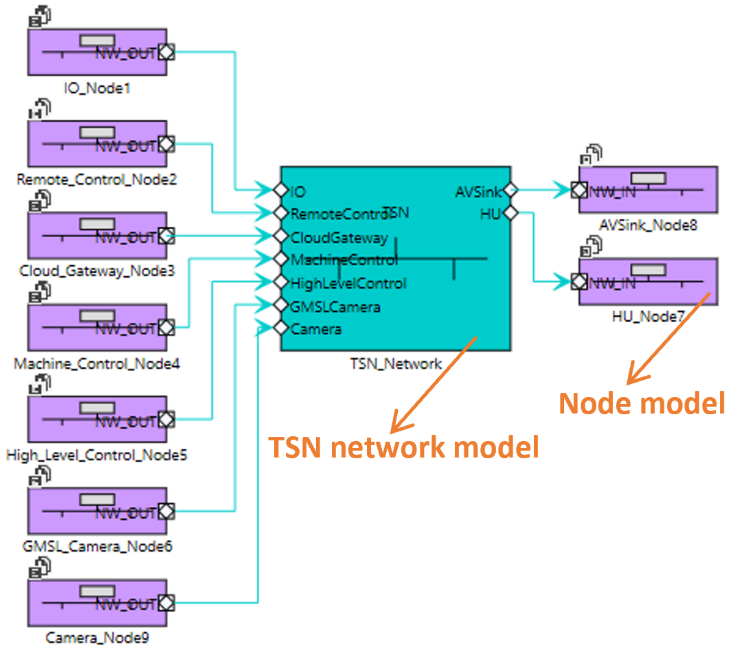


Figure 10.4. System-level software architecture of the use-case.

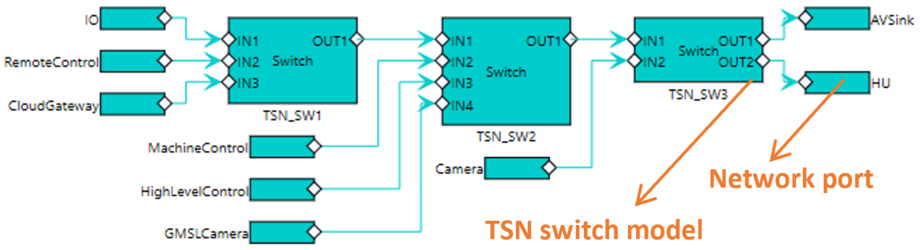


Figure 10.5. The TSN network model of the use-case.

timing model is required as a necessary input for the timing analysis engines. We also proposed a methodology for automated extraction of timing information for the instances of this model from the software architectures of the systems. Hence, the proposed methodology facilitates automated end-to-end data-propagation delay analysis of the software architectures of these systems. As a proof-of-concept, we implemented

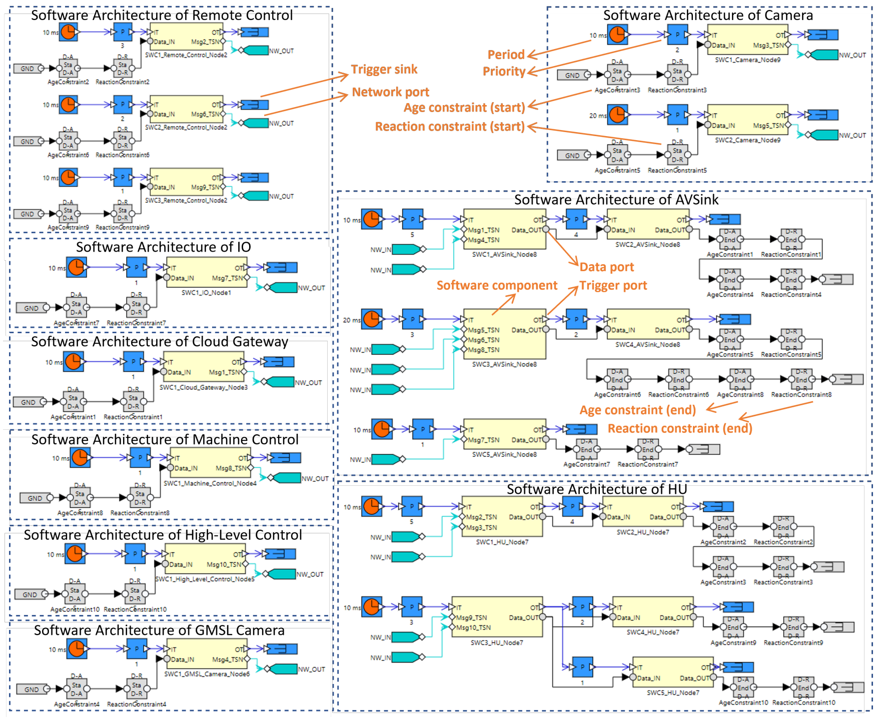

Software Architecture of Remote Control
Software Architecture of Camera
Software Architecture of IO
Software Architecture of Cloud Gateway
Software Architecture of Machine Control
Software Architecture of High-Level Control
Software Architecture of GMSL Camera
Software Architecture of AVSink
Software Architecture of HU
Trigger sink
Network port
Period
Priority
Age constraint (start)
Reaction constraint (start)
Data port
Software component
Trigger port
Age constraint (end)
Reaction constraint (end)

Figure 10.6. Software architecture of the nodes along with the timing constraints specified on ten distributed chains in RCM.

the end-to-end timing model and the model extraction methodology in an industrial tool suite, namely Rubus-ICE. We evaluated the proposed methodology using a vehicular industrial use-case. In this regard, we modelled the software architecture of the use case with an industrial component model (RCM) and performed its timing analysis using the proposed methodology in Rubus-ICE tool suite. The results demonstrate the applicability of the proposed model and methodology. The proposed methodology can be adapted for other component models that use the principles of model- and component-based software development and use the pipe-and-filter communication among software components such as AUTOSAR. One future research direction is to consider back-propagation of the analysis results for refining the software architecture and re-configuring the models of TSN networks.

Acknowledgment: This work is supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the DESTINE, INTERCONNECT and PROVIDENT projects and by the Swedish Knowledge Foundation via the projects DPAC & HERO. We would like to thank all industrial partners, in particular HIAB, Arcticus Systems and Volvo CE.

Table 10.1. Extracting the proposed timing model for TSN.

Component	Proposed Timing Model (c, Figure 10.2)	Proposed Parameters	User	SWA-d	Conf.	Analysis
Node	End-station ($c2$, Figure 10.2)	A$_{ij}$	✓			
		C$_{ij}$		✓		
SWC	Task ($c2$, Figure 10.2)	C_{ijk}	✓			
		T_{ijk}	✓	✓		
		O_{ijk}		✓	✓	
		P_{ijk}	✓			
		J_{ijk}				✓
		B_{ijk}				✓
		R_{ijk}				✓
		D_{ijk}		✓	✓	
Network	Network ($c3$, Figure 10.2)	s_i		✓		
		$\mathcal{L}_i$		✓		
		$\mathcal{J}_i$		✓		
		$Slope$		✓	✓	
		Preemption		✓	✓	
Message	Message ($c3$, Figure 10.2)	$\mathcal{X}_{ij}$		✓		
		C_{ij}		✓		
		P_{ij}		✓		
		$Size_{ij}$	✓			
		T_{ij}		✓		
		$\mathcal{L}_{ij}$	✓	✓	✓	
		J_{ij}				✓
		B_{ij}				✓
		R_{ij}				✓
		$\mathcal{O}_{ij}$		✓	✓	
		D_{ijk}		✓	✓	
SWC Chain	Transaction ($c4$, Figure 10.2)	Γ^d	✓			
		$Chain^d$	✓			
		T^d		✓		
Requirement	Requirement ($c5$, Figure 10.2)	Cr^d	✓		✓	
		a^d				✓
		r^d				✓

Table 10.2. Evaluation settings for the use-case based on the distributed chains (transactions). All times are in milliseconds.

Γ^d	Source ($\mathcal{E}_i$)	Source tasks (τ^d_{ijk}): [id, P_{ijk},C_{ijk}, T_{ijk}]	Message (m^d_{jk}): [id, P_{jk},$Size_{jk}$, T_{jk}, $\mathcal{O}^r_{jk}$]	Destination ($\mathcal{E}_i$)	Destination tasks (τ^d_{ijk}): [id, P_{ijk},C_{ijk}, T_{ijk}]	
		Task 1			Task 2	Task 3
1	Cloud Gateway (3)	[$\tau^1_{3,1,1}$,1,0.05,10]	[$m^1_{1,1}$, ST,1542,10,0.038]	AVSink (8)	[$\tau^1_{8,1,1}$,5,0.05,10]	[$\tau^1_{8,1,2}$,4,0.05,10]
2	Remote Control (2)	[$\tau^2_{2,1,1}$,3,0.05,10]	[$m^2_{1,2}$, ST,1542,10,0.038]	HU (7)	[$\tau^2_{7,1,1}$,5,0.05,10]	[$\tau^2_{7,1,2}$,4,0.05,10]
3	Camera (9)	[$\tau^3_{9,1,1}$,2,0.05,10]	[$m^3_{1,3}$, ST,1542,10,0.013]	HU (7)	[$\tau^3_{7,1,1}$,5,0.05,10]	[$\tau^3_{7,1,2}$,4,0.05,10]
4	GMSL Camera (6)	[$\tau^4_{6,1,1}$,1,0.05,10]	[$m^4_{1,4}$, ST,1542,10,0.025]	AVSink (8)	[$\tau^4_{8,1,1}$,5,0.05,10]	[$\tau^4_{8,1,2}$,4,0.05,10]
5	Camera (9)	[$\tau^5_{9,1,2}$,1,0.05,20]	[$m^5_{1,5}$, BE,1542,20,0]	AVSink (8)	[$\tau^5_{8,1,3}$,3,0.05,20]	[$\tau^5_{8,1,4}$,2,0.05,20]
6	Remote Control (2)	[$\tau^6_{2,1,2}$,2,0.05,10]	[$m^6_{1,6}$, BE,1542,10,0]	AVSink (8)	[$\tau^6_{8,1,3}$,3,0.05,20]	[$\tau^6_{8,1,4}$,2,0.05,20]
7	I/O (1)	[$\tau^7_{1,1,1}$,1,0.05,10]	[$m^7_{1,7}$, BE,1542,10,0]	AVSink (8)	[$\tau^7_{8,1,5}$,1,0.05,10]	
8	Machine Control (4)	[$\tau^8_{4,1,1}$,1,0.05,10]	[$m^8_{1,8}$, BE,1542,10,0]	AVSink (8)	[$\tau^8_{8,1,3}$,3,0.05,20]	[$\tau^8_{8,1,4}$,2,0.05,20]
9	Remote Control (2)	[$\tau^9_{2,1,3}$,1,0.05,10]	[$m^9_{1,9}$, BE,1542,10,0]	HU (7)	[$\tau^9_{7,1,3}$,3,0.05,10]	[$\tau^9_{7,1,4}$,2,0.05,10]
10	High-level Control (5)	[$\tau^{10}_{5,1,1}$,1,0.05,10]	[$m^{10}_{1,10}$, BE,1542,10,0]	HU (7)	[$\tau^{10}_{7,1,3}$,3,0.05,10]	[$\tau^{10}_{7,1,5}$,1,0.05,10]

Table 10.3. Calculated data age and reaction time delays, and message response times for each transaction.

Trans. (Γ^d)	$\Gamma^d.r^d$ (ms)	$\Gamma^d.a^d$ (ms)	$m_{ij}.R_{ij}$ (ms)
1	30.100	20.100	0.050
2	30.100	20.100	0.050
3	30.100	20.100	0.025
4	30.100	20.100	0.038
5	60.300	40.300	0.153
6	50.300	30.300	0.332
7	20.550	10.550	0.332
8	50.300	30.300	0.178
9	30.300	20.300	0.332
10	30.300	20.300	0.255

Bibliography

[1] L. Lo Bello, R. Mariani, S. Mubeen, S. Saponara. Recent Advances and Trends in On-Board Embedded and Networked Automotive Systems. *IEEE Transactions on Industrial Informatics*, November 2019.

[2] *802.1Q-2022 - IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks*. IEEE, 2022.

[3] IEEE. *IEEE Std. 802.1Qbv, IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks, Amendment 25: Enhancement for Scheduled Traffic*, 2015.

[4] IEEE. *IEEE Std. 802.1Qbu-2016: IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks, Bridges and Bridged Networks - Amendment 26: Frame Preemption*, 2016.

[5] M. Ashjaei, L. Lo Bello, M. Daneshtalab, G. Patti, S. Saponara, S. Mubeen. Time-Sensitive Networking in Automotive Embedded Systems: State-of-the-Art and Research Opportunities. *Journal of Systems Architecture*, August 2021.

[6] T. Vale, I. Crnkovic, E. S. de Almeida, P. A. da Mota Silveira Neto, Y. Cerqueira Cavalcanti, S. R. de Lemos Meira. Twenty-Eight Years of Component-Based Software Engineering. *Journal of Systems & Software*, 2016.

[7] T. A. Henzinger, J. Sifakis. The Embedded Systems Design Challenge. In *14th International Symposium on Formal Methods*, 2006.

[8] AUTOSAR Consortium, AUTOSAR Techincal Overview, Release 4.1, Rev.2, Ver.1.1.0., `http://autosar.org`.

[9] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, K. Lundback. The Rubus Component Model for Resource Constrained Real-Time Systems. In *IEEE Symposium on Industrial Embedded Systems*, 2008.

[10] C. Wolff , L. Krawczyk, R. Höttger, C. Brink, U. Lauschner, D. Fruhner, E. Kamsties, B. Igel. AMALTHEA — Tailoring Tools to Projects in Automotive Software Development. In *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, 2015.

[11] EAST-ADL Domain Model Specification, V2.1.12,. http://www.east-adl.-info/Specification/V2.1.12/EAST-ADL-Specification_V2.1.12.pdf.

[12] P.H. Feiler, D.P. Gluch, J.J. Hudak. The Architecture Analysis & Design Language (AADL): An introduction. Technical report, 2006.

[13] N. Feiertag, K. Richter, J. Nordlander, 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.

[14] M. Becker, D. Dasari, S. Mubeen, M. Behnam, T. Nolte. End-to-end Timing Analysis of Cause-Effect Chains in Automotive Embedded Systems. *Journal of Systems Architecture*, 2017.

[15] S. Mubeen, T. Nolte, M. Sjödin, J. Lundbäck, K. Lundbäck. Supporting Timing Analysis of Vehicular Embedded Systems through the Refinement of Timing Constraints. *International Journal on Software and Systems Modeling*, 2017.

[16] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, R. Ernst. System-level Performance Analysis - the SymTA/S Approach. *Computers and Digital Techniques*, 2005.

[17] S. Mubeen, J. Mäki-Turja, M. Sjödin. Support for End-to-End Response-Time and Delay Analysis in the Industrial Tool Suite: Issues, Experiences and a Case Study. *Computer Science and Information Systems*, January 2013.

[18] S. Mubeen, H. Lawson, J. Lundbäck, M. Gålnander, K. Lundbäck. Provisioning of Predictable Embedded Software in the Vehicle Industry: The Rubus Approach. In *4th International Workshop on Software Engineering Research and Industry Practice*, 2017.

[19] S. Mubeen, M. Ashjaei, M. Sjödin. Holistic Modeling of Time Sensitive Networking in Component-based Vehicular Embedded Systems. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2019.

[20] A. Arestova, M. Martin, K-S J. Hielscher, R. German. A Service-oriented Real-time Communication Scheme for AUTOSAR Adaptive using OPC UA and Time-sensitive Networking. *Sensors*, 2021.

[21] A. Bucaioni, L. Addazi, A. Cicchetti, F. Ciccozzi, R. Eramo, S. Mubeen, M. Sjödin. MoVES: A Model-Driven Methodology for Vehicular Embedded Systems. *IEEE Access*, 2018.

[22] A. Bucaioni, M. Becker, J. Lundbäck, H. Mackamul. From AMALTHEA to RCM and Back: a Practical Architectural Mapping Scheme. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2020.

[23] A. Bucaioni, V. Dimic, H. Lönn, M. Gålnander, J. Lundbäck. Transferring a Model-based Development Methodology to the Automotive Industry. In *22nd IEEE International Conference on Industrial Technology*, 2021.

[24] S. Mubeen, M. Gålnander, J. Lundbäck, K. Lundbäck. Extracting Timing Models from Component-based Multi-criticality Vehicular Embedded Systems. In *15th International Conference on Information Technology : New Generations*, 2018.

[25] ISO 11898-1. Road Vehicles–Interchange of Digital Information–Controller Area Network (CAN) for High-speed Communication, ISO Standard-11898, November 1993.

[26] CANopen Application Layer and Communication Profile. CiA Draft Standard 301. Version 4.02. February 13, 2002. http://www.can-cia.org/index.php?id=440.

[27] AUTOSAR Requirements on Communication, Release 4.2.1. `www.autosar.org`, accessed on March 15, 2019.

[28] B. Houtan, M-O. Aybek, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Mubeen. End-to-end Timing Model Extraction from TSN-Aware Distributed Vehicle Software. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2022.

[29] F. Stappert, J. Å. Jönsson, J. Mottok, R. Johansson. A Design Framework for End-To-End Timing Constrained Automotive Applications. In *Embedded Real Time Software and Systems Conference*, 2010.

[30] M. Becker, Matthias, D. Dasari, S. Mubeen, M. Behnam, T. Nolte. Synthesizing Job-level Dependencies for Automotive Multi-rate Effect Chains. In *2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications*, 2016.

[31] M. Ashjaei, N. Khalilzad, S. Mubeen, M. Behnam, I. Sander, L. Almeida, T. Nolte. Designing End-to-end Resource Reservations in Predictable Distributed Embedded Systems. *Real-Time Sys.*, 2017.

[32] S. Mubeen, J. Mäki-Turja, M. Sjödin. Communications-Oriented Development of Component-Based Vehicular Distributed Real-Time Embedded Systems. *Journal of Systems Architecture*, 2014.

[33] M. Ashjaei, S. Mubeen, J. Lundbäck, M. Gålnander, K-L. Lundbäck, T. Nolte. Modeling and Timing Analysis of Vehicle Functions Distributed over Switched Ethernet. In *43rd Annual Conference of the IEEE Industrial Electronics Society*, 2017.

[34] S. Mubeen, M. Gålnander, A. Bucaioni, J. Lundbäck, K-L. Lundbäck. Timing Verification of Component-based Vehicle Software with Rubus-ICE: End-user's Experience. In *IEEE/ACM 1st International Workshop on Software Qualities and their Dependencies*, 2018.

[35] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Mubeen. Supporting End-to-end Data-propagation Delay Analysis for TSN Networks. Technical report, 2021.

[36] J.C. Palencia, M. Gonzalez Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. *Proceedings 19th IEEE Real-Time Systems Symposium*, 1998.

[37] A. Minaeva, Z. Hanzálek. Survey on Periodic Scheduling for Time-Triggered Hard Real-Time Systems. *Association for Computing Machinery*, 2022.

[38] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Afshar, S. Mubeen. Schedulability Analysis of Best-effort Traffic in TSN Networks. In *26th IEEE International Conference on Emerging Technologies and Factory Automation*, 2021.

[39] J. Cao, M. Ashjaei, P. J. Cuijpers, R. J. Bril, J. Lukkien. Independent Yet Tight WCRT Analysis for Individual Priority Classes in Ethernet AVB. In *International Workshop on Factory Communication Systems*, 2018.

[40] Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Saad Mubeen. Synthesising Schedules to Improve QoS of Best-effort Traffic in TSN Networks. In *29th International Conference on Real-time Networks and Systems*, 2021.

[41] F. Pozo, G. Rodriguez-Navas, H. Hansson. Schedule Reparability: Enhancing Time-triggered Network Recovery upon Link Failures. In *IEEE International Conference on Embedded and Real-time Computing Systems and Applications*, 2018.

[42] V. Gavriluţ, L. Zhao, M. L. Raagaard, P. Pop. AVB-aware Routing and Scheduling of Time-triggered Traffic for TSN. *IEEE Access*, 2018.

[43] S. Lindner M. Menth T. Stüber, L. Osswald. A Survey of Scheduling in Time-Sensitive Networking (TSN). *arXiv preprint arXiv:2211.10954*, 2022.

Chapter 11

Paper F: Bandwidth Reservation Analysis for Schedulability of AVB Traffic in TSN

Bahar Houtan, Mohammad Ashjaei, Masoud Daneshtalab, Mikael Sjödin, Saad Mubeen.
In the 25^{th} IEEE International Conference on Industrial Technology.

Abstract

In this paper, we present a bandwidth reservation analysis for Audio-Video Bridging (AVB) traffic in the Time-Sensitive Networking (TSN) standards. The proposed analysis is based on the existing worst-case response-time analysis and can be used to calculate the minimum required bandwidth for guaranteeing the schedulability of messages in AVB classes. The proposed analysis allocates per-link bandwidth to AVB traffic that is sufficient to ensure its schedulability when a combination of the Credit-Based Shaper and Time-Aware Shaper mechanisms are used. We evaluate the proposed analysis using an automotive industrial use case. We evaluate the schedulability of AVB traffic by comparing the proposed analysis with the utilization-based bandwidth reservation as recommended by the TSN standards.

11.1 Introduction

The set of Time-Sensitive Networking (TSN) [1] standards introduce various mechanisms, such as time synchronization, Time-Aware shaper (TAS), and Credit-Based shaper (CBS) to switched Ethernet. The co-existence of these mechanisms facilitates the network communication design for highly predictable distributed embedded systems. However, combining these mechanisms may introduce complications in both design and analysis. Besides, a significant challenge in realizing such systems is guaranteeing the timing requirements of traffic. To resolve this, there are system and network design recommendations for using the TSN standards. However, the recommendations are generalized and do not specifically address all use cases of the standards. In this paper, we focus on the recommendations of the standards for the operation of CBS [1] for Audio-Video Bridging (AVB) classes. Particularly, the standards suggest setting up the credit of the AVB classes according to the available bandwidth in each class [1]. Previous works, such as [2] and [3], proved that the bandwidth allocation strategy based on the standard recommendation results in under-reservation of the bandwidth for AVB traffic which limits AVB traffic to the extent that they miss their deadlines. Hence, this set of works proposes techniques for calculating the minimum required bandwidth for AVB traffic, while guaranteeing the schedulability of all traffic in these classes. The works show that the bandwidth allocation recommended by the standard is not enough for scheduling the AVB traffic in the network.

The existing literature addresses the aforementioned problem with different analysis approaches. For example, worst-case Response Time Analysis (RTA) is used in [2] to calculate the minimum required bandwidth for CBS in a non-preemptive combination of TAS and CBS, i.e., in AVB ST[4]. AVB ST was proposed before the standard amendments on the support for Scheduled Traffic (ST), thus the model and credit utilization were different than the standards. In addition, there are a few works that use eligible intervals, such as [3] and [5], to analyze the bandwidth reservation for TSN networks. However, these works are limited to a single-switch architecture and the effect of ST traffic is neglected.

As a result, to the best of our knowledge, there is no solution to analyze the required bandwidth for AVB traffic classes in TSN networks considering features such as multi-hop architecture, ST traffic effects, and preemption support of ST over lower-priority traffic classes. Therefore, in this paper, we fill this gap by proposing a Bandwidth Reservation Analysis (BRA) for calculating the lowest value of the bandwidth for classes A and B such that they become schedulable. The solution is based on the existing RTA [6] to consider the impact of ST traffic, multi-hop architecture, and preemption of ST on the lower-priority traffic.

11.2 Background and Related Work

11.2.1 Time-Sensitive Networking (TSN) Standards

TAS as depicted in Figure 11.1 uses the time-stamped gate states defined in the Gate Control List (GCL) for deterministic transmission of ST. The rows of GCL indicate the time and order of enabling the gates connected to the queues in the TSN switch. Figure 11.1 shows a simplified example of TSN egress port with four queues, however, there can be a maximum of eight queues that can be controlled by eight-bit rows of GCL. ST queues preempt the transmission of lower-priority traffic. CBS reserves bandwidth for the AVB classes while it also limits the AVB traffic to interfere with traffic in the lower-priority classes. CBS can be enabled for all of the eight available queues per port of a TSN switch in the standards [1], thus it can include maximum eight classes. It is

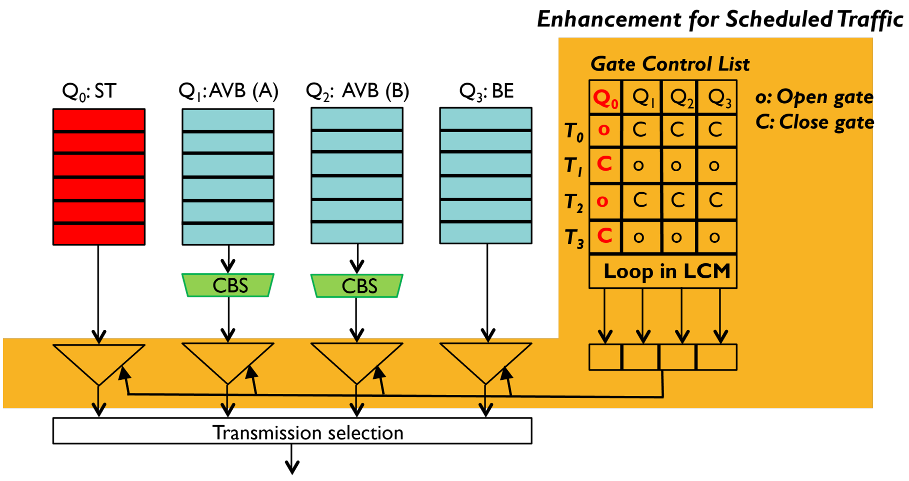


Figure 11.1. TSN egress port.

very common to use only classes A and B in analysis and practice. In such examples in the presence of TAS mechanism, the ST class represents the higher-priority preemptive class, class A represents the higher-priority AVB class, class B represents the medium priority AVB class, and Best-Effort (BE) traffic represents the lowest priority traffic class. Each AVB class has a specific credit associated to it. Transmission of traffic in these classes is determined based on the credit states explained in the following:

- ***Positive credit***: message in an AVB class starts its transmission when it is ready for transmission and the credit of the class is greater than or equal to zero. If at the same time there are other active sources of interference and blocking using the bandwidth, the message must wait. The credit increases while this message waits for available bandwidth.

- ***Frozen credit***: ST message preempts AVB traffic. During preemption, the credit freezes until the transmission of the ST message is completed.
- ***Negative credit***: the credit decreases while a message in this class uses the bandwidth. If a message is activated while the credit is below zero, it will not be transmitted. The message must wait until the credit is replenished to zero, or goes above zero, i.e., until the bandwidth is free for its transmission. When there are no messages ready for transmission in the class and the credit is negative, the credit increases until reaches zero.

11.2.2 Related Works

The work in [2] derives minimum credits for schedulability of the classes A and B by taking advantage of RTA. The system model in this work is based on the definition of protected windows in AVB ST as given in [4]. AVB ST did not support ST preemption on the lower-priority traffic. Furthermore, the work in [3] proposed a strategy to set a lower bound of credits for the schedulability of AVB traffic. However, the work still lacks support for all existing TSN mechanisms, such as preemption and ST transmission. Also, the work considers only a single-switch architecture. Furthermore, the work in [5] extends the aforementioned approach and proposes two new metrics as the relation between the credits of different AVB classes in a single switch system model. In [7], the authors discuss that the over-reservation of bandwidth does not necessarily lead to lower end-to-end delays and/or schedulability of streams. As it can be seen from above there are very few works addressing the problem of AVB credit allocation to guarantee the schedulability of AVB traffic in TSN networks.

In this paper, we consider the definitions of the TAS, CBS and strict priority mechanism according to the TSN standards [1] and the latest improvements of RTA in [6, 8]. We mathematically propose a solution to find the lowest value for credits of classes A and B such that they become schedulable. Such a solution was missing in the state of the art which can help the designers to set the credits for AVB traffic classes.

11.3 System Model

In this paper, we consider TSN classes ST, AVB (A and B), and BE. The total set of links in the network topology is represented as $\mathcal{L}$, and l denotes the ID of a link within the set. A link represents a one-directional connection between an end station to a switch or a switch to another switch. Network utilization of each traffic class on a specific link is shown by $U_{z,l}$, where z denotes the traffic class. The notations $\alpha^{+}_{z,l}$ and $\alpha^{-}_{z,l}$ are the idle slope and send slope values associated with the link l and z points to an AVB class. The send slope is the rate at which the credit of an AVB class is consumed, and the idle slope is the rate at which the credit increases. R is the network capacity, which is the number of bits transmitted in a unit of time ($Mbps$). The available bandwidth ratio on link l is denoted by BW_l. The sum of the idle slope and send slope of each AVB class equals the available bandwidth ratio ($\alpha^{+}_{z,l} + \alpha^{-}_{z,l} = BW_l$). For example, if credit of class z is $\alpha^{+}_{z,l} = 0.5$, the available bandwidth ratio is $BW_l = 1$, and the network capacity is $R = 100Mbps$, then class z can use $50Mbps$ of the network capacity. Finally, the delay due to the hardware design of the switch is bounded by γ. The traffic is represented by a set of messages $\mathcal{M}$, and m_i denotes a message within this set.

A message models a stream of data that is transmitted between a source end-station and a destination end-station and consists of the attributes as shown in the tuple in Eq. (11.1):

$$m_i := \left\langle T_i, D_i, C_i, P_i, Size_i, \mathcal{L}_i, \mathcal{O}_i, J^l_i, RT^l_i \right\rangle . \tag{11.1}$$

The message m_i is periodically activated for transmission with the period T_i. The deadline of m_i is denoted by D_i. We consider implicit deadlines, i.e., the deadline of each message is equal to its period. Besides, on link l in the route of the message, we can set a local deadline for m_i denoted by D^l_i. The deadline of a message is equal to the summation of all local deadlines of this message. The message priority is represented by P_i that can obtain a value from the set of classes $\{ST, A, B, BE\}$. We consider only two AVB classes (classes A and B). In this paper, we assume class ST has the highest priority in TSN. The class ST preempts the traffic in the other lower-priority classes. AVB and BE are preempted by the ST class (i.e., lower-priority classes are preemptable). Moreover, express queue is not enabled for AVB and BE thus they cannot preempt their lower-priority classes (i.e., non-preemptive transmission). The notation $\mathcal{O}_i$ represents the set of offsets of an ST message m_i within $\mathcal{L}_i$, where $\mathcal{L}_i$ is the set of all links in the route of m_i from the source end-station to the destination end-station, and $\mathcal{L}_i \subset \mathcal{L}$.

The transmission time of m_i is represented by C_i which is calculated based on the network capacity R and the payload size ($Size_i$) of the message, according to $((Size_i + OH) * 8)/R$. The parameter OH is the size of a TSN (Ethernet) frame header in Bytes added to the m_i's payload size. The transmission time of OH is denoted by v in the system model. The fragments of a preempted message obtain an individual TSN (Ethernet) frame header. According to the IEEE 802.3br standard, a message fragment that is less than 123 $Bytes$ cannot be preempted. A guard band is a reserved slot added before the ST schedule slots to prevent any blocking by a lower-priority message.

This is for the case when a lower-priority message cannot be preempted because it is activated before a higher-priority message (i.e., ST). The duration of the guard band is represented by λ. The delay variations in the transmission of the message m_i on the link l is denoted by the queuing jitter J_i^l. The worst-case response time of m_i of class A and B on link l is calculated by analysis and represented by RT_i^l.

11.4 Response-time Analysis for TSN

This section revisits the existing RTA for classes A and B in TSN, presented in [6], which is the basis for this paper's proposed BRA. The sources of interference and blocking that contribute to the worst-case response time of a message under analysis include: (1) interference from the messages of the higher-priority traffic (hp); (2) interference from the messages of the same-priority traffic (sp); and (3) blocking from the messages of the lower-priority traffic (lp). In this section, a message under analysis is denoted by m_i. A message is schedulable if its worst-case response time (RT_i) is less than its deadline (D_i).

11.4.1 Response-time analysis for class A messages

Let m_i be the class A message under analysis. m_i can experience interference from higher-priority messages belonging to the ST class (hp), same-priority messages from class A (sp), and blocking from one of the lower-priority classes B or BE [6].

Higher-priority interference

The ST traffic is scheduled offline using existing scheduling approaches (more details in [9]). RTA in this paper adapts offset-based analysis in [10] for calculation of preemption of m_i by the ST traffic. The set of instances of an ST message m_j on link l is indicated by $\mathcal{I}_j^l$ and is extracted by their offsets according to Eq. (11.2).

$$\mathcal{I}_j^l = \{(k-1)T_j + O_j^l - \lambda : k = 1..n, n = \frac{T_{lcm}}{T_j}\} \tag{11.2}$$

In the offset-based analysis, all ST messages are included in a transaction, as shown in the example Figure 11.2. In this example, there are two ST messages, namely m_x and m_w. The notation T_{lcm} is the hyper period of the ST transaction, i.e., the least common multiple (LCM) of the periods of all ST messages in the transaction. In the example in Figure 11.2, the T_{lcm} includes one instance of m_x and two instances of m_w. The worst-case interference from the ST traffic on m_i is when m_i is activated at the same

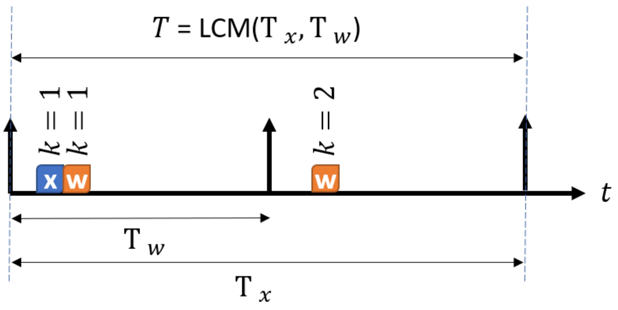


Figure 11.2. An example of a transaction [8].

time as an instance of one of the ST messages in the transaction. We call this point in time a critical instant for m_i. There can be several instances of ST messages in the transaction. Thus there can be several critical instant combinations for the interference of ST on m_i. Critical instant combinations are formed with regards to the activation pattern of instances of one ST message in the transaction (i.e. m_c or critical instant candidate). $\mathfrak{I}_c^l$ is the set of activation times of the instances of m_c within the hyper period. Eq. (11.3) calculates the phasing of the instances of the ST message m_j on link l denoted by $\Phi_{jc[k]}^l$ according to the instance k of the critical instant candidate m_c.

$$\Phi_{jc[k]}^l = (O_j^l - \mathfrak{I}_c^l[k]) \ mod \ T_{lcm} \tag{11.3}$$

The higher-priority interference received by m_i, denoted by $W_{c[k]}^l(t)$, includes preemption by the messages in class ST as shown in Eq. (11.4).

$$W_{c[k]}^l(t) = \sum_{\forall j \in ST \ \wedge l \in \mathcal{L}_j} \left(\left\lfloor \frac{\Phi_{jc[k]}^l}{T_{lcm}} \right\rfloor + \left\lceil \frac{t - \Phi_{jc[k]}^l}{T_{lcm}} \right\rceil \right) C_j \tag{11.4}$$

Similarly, the preemption overheads, denoted by $V_{c[k]}^l(t)$, are calculated as shown in Eq. (11.5). For more details on preemption overhead and interference calculations, we refer the readers to [6].

$$V_{c[k]}^l(t) = \sum_{\forall j \in ST \ \wedge l \in \mathcal{L}_j} \left(\left\lfloor \frac{\Phi_{jc[k]}^l}{T_{lcm}} \right\rfloor + \left\lceil \frac{t - \Phi_{jc[k]}^l}{T_{lcm}} \right\rceil \right) v \tag{11.5}$$

Same-priority interference and blocking by lower priority

To account for the same-priority interference and the blocking for m_i, we follow the lemma introduced in [6]. Accordingly, the worst-case blocking for m_i is caused by an instance of the lower-priority message that has the maximum transmission time, as shown in Eq. (11.7), provided that the other same-priority messages interfere with m_i with the inflation factor $(1 + \frac{\alpha^-_{A,l}}{\alpha^+_{A,l}})$. The same-priority interference on message m_i, denoted by ISA^l_i, is calculated according to Eq. (11.6).

$$ISA^l_i = \sum_{\substack{\forall m_j \in sp(m_i), i \neq j \\ \wedge\ l \in \mathcal{L}_j}} \left(1 + \frac{\alpha^-_{A,l}}{\alpha^+_{A,l}}\right) C_j \tag{11.6}$$

where, the blocking on m_i by the lower-priority messages on link l, denoted by BB^l_i, is obtained by selecting the transmission time of a lower-priority message with the maximum frame size, as shown in Eq. (11.7).

$$BB^l_i = \max_{\substack{\forall m_j \in lp(m_i) \\ \wedge\ l \in \mathcal{L}_j}} \{C_j\} \tag{11.7}$$

Worst-case response time of class A

The worst-case response time of m_i on link l according to a critical instant candidate m_c is derived by the iterative equation Eq. (11.8).

$$RT^{l,(x+1)}_{ic[k]} = W^l_{c[k]}(RT^{l,(x)}_{ic[k]}) + V^l_{c[k]}(RT^{l,(x)}_{ic[k]}) + BB^l_i + ISA^l_i + C_i \tag{11.8}$$

The worst-case response time is denoted by $RT^l_{ic[k]}$, where the index of the current and the next iterations are specified by (x) and $(x + 1)$ superscripts, respectively (iteration starts at $x = 0$). The iteration stops when $RT^l_{ic[k]}$ converges to a single value, i.e., $RT^{l,(x)}_{ic[k]} = RT^{l,(x+1)}_{ic[k]}$. Additionally, the analysis stops when the last calculated $RT^l_{ic[k]}$ is higher than the deadline of m_i. Finally, the worst-case response time of m_i on link l is the maximum among all the response times calculated for this message based on all the critical instant combinations, as shown in Eq. (11.9).

$$RT^l_i = \max_{\substack{\forall m_c \in \{classST\} \\ \wedge \forall k \in \mathcal{I}^l_c}} \{RT^l_{ic[k]}\} \tag{11.9}$$

11.4.2 Response-time analysis for class B messages

Now let m_i be the class B message under analysis. The worst-case response time of m_i occurs when it is interfered by the higher-priority classes (hp), i.e., class ST and class A; and the other same-priority messages in class B (sp). Moreover, m_i receives worst-case blocking from the lower-priority class BE message [6].

Higher-priority interference

There are two higher-priority classes for class B, which have different types of interference on m_i. Firstly, interference from the class ST on class B messages is by preemption, which is calculated the same way as presented in Section 11.4.1 for class A. Secondly, the interference by class A happens when m_i needs to wait for the higher-priority messages from class A to finish their transmission. The interference from class A, denoted by $IA_i^l(\omega_{ic[k]}^l(q))$, is calculated according to Eq. (11.10).

$$IA_i^l(\omega_{ic[k]}^l(q)) = \sum_{\substack{\forall m_j \in classA, i \neq j \\ \wedge\ l \in \mathcal{L}_j}} \left\lfloor \frac{\omega_{ic[k]}^l(q) + J_j^l}{T_j} + 1 \right\rfloor C_j \tag{11.10}$$

The interference by the class A messages on the instance q of m_i is updated with the last value of the busy period ($\omega_{ic[k]}^l(q)$) which is calculated for this instance according to the k^{th} instance of m_c in class ST. The worst-case response time can be for any of the q instances of m_i, therefore all the q instances must be taken into account. For more information and the proofs, we refer readers to [6].

Same-priority interference and blocking by lower priority

The same-priority interference of other class B messages on instance q of m_i, denoted by $ISB_i^l(q)$, is calculated by Eq. (11.11).

$$ISB_i^l(q) = \sum_{\substack{\forall m_j \in sp(m_i), i \neq j \\ \wedge l \in \mathcal{L}_j}} \left(1 + \frac{\alpha_{B,l}^-}{\alpha_{B,l}^+}\right) \left\lfloor \frac{(q-1)T_i}{T_j} + 1 \right\rfloor .C_j \tag{11.11}$$

Similar to the analysis for class A (presented in Section 11.4.1), the other messages of the same-priority interfere with m_i by the inflation factor $\left(1 + \frac{\alpha_{B,l}^-}{\alpha_{B,l}^+}\right)$, provided that the largest message with lower priority is considered as the source of blocking.

Lower-priority blocking on link l is denoted by BBE_i^l, as shown in Eq. (11.12).

$$BBE_i^l = \max_{\substack{\forall m_j \in lp(m_i) \\ \wedge\ l \in \mathcal{L}_j}} \{C_j\} \tag{11.12}$$

Worst-case response time of class B

The worst-case response time of m_i in class B is performed in two phases. Firstly, we are interested in calculating the maximum length of the busy period, and for that, we need to carry out the analysis on q number of instances of m_i. The busy period for the q^{th} instance of m_i according to a critical instant candidate m_c is denoted by $\omega_{ic[k]}^l(q)$ and is calculated using Eq. (11.13).

$$\begin{aligned} \omega_{ic[k]}^{l,(x+1)}(q) = W_{c[k]}^l(\omega_{ic[k]}^{l,(x)}(q)) + V_{c[k]}^l(\omega_{ic[k]}^{l,(x)}(q)) + \\ IA_i^l(\omega_{ic[k]}^{l,(x)}(q)) + BBE_i^l + ISB_i^l(q) + (q-1)C_i \end{aligned} \tag{11.13}$$

The length of the busy period is updated until Eq. (11.13) converges (i.e., the newly computed values are equal to the previously computed value according to $\omega_{ic[k]}^{l,(x)}(q) = \omega_{ic[k]}^{l,(x+1)}(q)$).

In the second phase, the effect of interference by the preemption of the ST class on the response time of the instance q of m_i is taken into account. This phase of analysis, receives the worst-case busy period from the first phase of RTA, in Eq. (11.13). The response time of the q^{th} instance of m_i according to preemptions by the critical instant candidate based on the k^{th} instance of m_c is denoted by $RT_{ic[k]}^{l,(x+1)}(q)$ and is calculated using Eq. (11.14).

$$\begin{aligned} RT_{ic[k]}^{l,(x+1)}(q) = \omega_{ic[k]}^l(q) + W_{c[k]}^l(RT_{ic[k]}^{l,(x)}(q)) + \\ V_{c[k]}^l(RT_{ic[k]}^{l,(x)}(q)) + C_i - (q-1)T_i \end{aligned} \tag{11.14}$$

We calculate the maximum response time among all $RT_{ic[k]}^l$ of all q instances of m_i. Furthermore, the worst-case response time of m_i (RT_i^l) is the maximum $RT_{ic[k]}^l$ which is derived after considering all the critical instant combinations, as shown in Eq. (11.15).

$$RT_i^l = \max_{\substack{\forall c \in \{classST\} \wedge \\ \forall k \in \mathcal{I}_c^l}} \{ \max_{q=1..q_{max}} \{RT_{ic[k]}^l(q)\}\} \tag{11.15}$$

The maximum number of instances of m_i to be considered in RTA, denoted by q_{max}, is equal to the maximum value of q that satisfies Eq. (11.16):

$$\begin{aligned} &W^l_{c[k]}(\omega^l_{ic[k]}(q)) + V^l_{c[k]}(\omega^l_{ic[k]}(q)) + BBE^l_i + ISB^l_i \\ &+ \sum_{\substack{\forall m_j \in classA, \\ i \neq j \wedge l \in \mathcal{L}_j}} \left\lceil \frac{\omega^l_{ic[k]}(q) + j^l_j}{T_j} \right\rceil .C_j + q.C_i \leq q.T_i \end{aligned} \tag{11.16}$$

Eqs. (11.9) and (11.15) present the worst-case response time of class A and B on link l, respectively. The total response time for m_i crossing all links in its route is the summation of per-link response times.

11.5 Proposed bandwidth reservation analysis

In this section, we use RTA presented in Section 11.4 to propose an analysis for the bandwidth reservation of AVB messages in TSN. The proposed analysis considers preemption of the ST traffic on the lower-priority traffic according to the TSN standards [1], where class ST preempts lower-priority classes. The proposed analysis computes the lowest value of the credit (minimum idle slope, i.e., $\alpha^+_{z,l}$) which makes the system schedulable.

11.5.1 Bandwidth reservation analysis for class A

We define the local deadline D^l_i by which we desire the completion of m_i on link l. The credit of class A on link l must be sufficient for m_i to complete before D^l_i, i.e., $RT^l_i \leq D^l_i$. Reserving the bandwidth of class A according to the utilization of class A cannot guarantee that all messages in this class meet their deadlines. Therefore, class A requires more bandwidth than recommended by the standards [1]. Besides, the bandwidth for class A must not exceed the remaining bandwidth after the reserved slots for class ST, i.e., $\alpha^+_{A,l} \leq BW_l - U_{ST,l}$. The bandwidth of class A can be extracted from the inflation factor as shown in Eq. (11.17):

$$\left(1 + \frac{\alpha^-_{A,l}}{\alpha^+_{A,l}}\right) = \frac{\alpha^+_{A,l} + \alpha^-_{A,l}}{\alpha^+_{A,l}} = \frac{BW_l}{\alpha^+_{A,l}}. \tag{11.17}$$

The send slope is the difference between the bandwidth and the idle slope, i.e., $\alpha^-_{A,l} = BW_l - \alpha^+_{A,l}$. Thus for simplification, the same-priority interference (ISA^l_i) in Eq. (11.6) is reformulated to Eq. (11.18):

$$ISA^l_i = \frac{BW_l}{\alpha^+_{A,l}} \left(\sum_{\substack{\forall m_j \in sp(m_i), i \neq j \\ \wedge \, l \in \mathcal{L}_j}} C_j \right). \tag{11.18}$$

In Eq. (11.19), we rewrite Eq. (11.8) according to the term ISA_i^l in the analysis assuming that the worst-case response time is equal to the deadline, i.e., $RT_i^l = D_i^l$.

$$ISA_i^l = D_i^l - (W_{c[k]}^l(D_i^l) + V_{c[k]}^l(D_i^l) + BB_i^l + C_i) \tag{11.19}$$

Consequently, Eq. (11.18) and Eq. (11.19) are combined as Eq. (11.20):

$$\frac{BW_l}{\alpha_{A,l}^+}\left(\sum_{\substack{\forall m_j \in sp(m_i), i \neq j \\ \wedge\ l \in \mathcal{L}_j}} C_j\right) = D_i^l - (W_{c[k]}^l(D_i^l) + V_{c[k]}^l(D_i^l) + BB_i^l + C_i). \tag{11.20}$$

As shown in Eq. (11.20), various combinations of the critical instant candidates are involved in the analysis of the bandwidth for class A. Therefore, Eq. (11.20) leads to various idle slopes for m_i for each critical instant combination. The idle slope of m_i on link l for critical instant combination based on instance k of m_c at the local deadline is presented as $\alpha_{A,l}^+(\mathcal{I}_c^l[k], D_i^l)$ and is calculated according to Eq. (11.21).

$$\alpha_{A,l}^+(\mathcal{I}_c^l[k], D_i^l) = \left(\frac{\sum\limits_{\substack{\forall m_j \in sp(m_i), i \neq j \\ \wedge\ l \in \mathcal{L}_j}} C_j}{D_i^l - (W_{c[k]}^l(D_i^l) + V_{c[k]}^l(D_i^l) + BB_i^l + C_i)}\right) BW_l \tag{11.21}$$

The maximum idle slope is given by Eq. (11.22) for all the messages in class A:

$$\alpha_{A,l}^+ = \max_{\substack{\forall i \in \{classA\} \wedge\ l \in \mathcal{L}_i \\ \forall c \in \{classST\} \wedge\ \forall k \in \mathcal{I}_c^l}} \{\alpha_{A,l}^+(\mathcal{I}_c^l[k], D_i^l)\}. \tag{11.22}$$

The final idle slope given by Eq. (11.22) is applicable for all messages in this class to meet their deadlines. Hence, it can be set as the minimum credit that makes all class A messages schedulable.

11.5.2 Bandwidth reservation analysis for class B

The minimum bandwidth for schedulability of class B can be retrieved by reversing RTA. Since class B has lower priority than class ST and class A, calculating the minimum credit for this class is constrained to the remaining bandwidth after transmission of higher-priority classes, as shown in Eq. (11.23):

$$\alpha_{B,l}^+ \leq BW_l - (U_{ST,l} + \alpha_{A,l}^+). \tag{11.23}$$

Moreover, the busy period of the instance q of the message m_i according to the instance k of the critical instant candidate m_c is extracted from the second phase of RTA for class B (Eq. (11.14) presented in Section 11.4.2) which is denoted by $\omega^l_{ic[k]}(q)$ and is shown in Eq. (11.24).

$$\begin{gathered}\omega^l_{ic[k]}(q) = RT^{l,(x+1)}_{ic[k]}(q) - W^l_{c[k]}(RT^{l,(x)}_{ic[k]}(q)) - \\ V^l_{c[k]}(RT^{l,(x)}_{ic[k]}(q)) - C_i + (q-1)T_i\end{gathered} \tag{11.24}$$

Eq. (11.24) is rewritten as Eq. (11.25) with the assumption that in the worst-case the second phase of RTA for class B for analyzing instance q of m_i is converged at the local deadline of the message on the link l, i.e., $RT^l_{ic[k]}(q) \leq D^l_i$.

$$\omega^l_{ic[k]}(D^l_i, q) = D^l_i - W^l_{c[k]}(D^l_i) - V^l_{c[k]}(D^l_i) - C_i + (q-1)T_i \tag{11.25}$$

Then, the maximum busy period of m_i among all busy periods is denoted by $\omega^l_i(D^l_i, q)$ and is calculated as shown in Eq. (11.26).

$$\omega^l_i(D^l_i, q) = \max_{\forall c \in \{classST\} \wedge \forall k \in \mathcal{I}^l_c} \{\omega^l_{ic[k]}(D^l_i, q)\} \tag{11.26}$$

Similar to the inflation factor of class A, as shown in Eq. (11.17), the inflation factor for class B can be presented as $\frac{BW_l}{\alpha^+_{B,l}}$. The contribution of the same-priority interference in class B into the busy period is denoted by $ISB^l_i(q)$ and is rewritten by inserting the new form of inflation factor of class B to Eq. (11.11) as shown in Eq. (11.27).

$$ISB^l_i(q) = \frac{BW_l}{\alpha^+_{B,l}} \left(\sum_{\substack{\forall m_j \in sp(m_i), i \neq j \\ \wedge l \in \mathcal{L}_j}} \left\lfloor \frac{(q-1)T_i}{T_j} + 1 \right\rfloor .C_j \right) \tag{11.27}$$

For readability, we denote part of Eq. (11.27) which is in brackets by $Y(q) = \sum_{\substack{\forall m_j \in sp(m_i), i \neq j \\ \wedge l \in \mathcal{L}_j}} \left\lfloor \frac{(q-1)T_i}{T_j} + 1 \right\rfloor .C_j$ and reformulate Eq. (11.27) to Eq. (11.28).

$$ISB^l_i(q) = \frac{BW_l}{\alpha^+_{B,l}} * Y(q) \tag{11.28}$$

Furthermore, the first phase of RTA for class B is reformulated based on the term for same-priority interference ($ISB^l_i(q)$), as shown in Eq. (11.29).

$$\begin{gathered}ISB^l_i(q) = \omega^{l,(x+1)}_{ic[k]}(q) - W^l_{c[k]}(\omega^{l,(x)}_{ic[k]}(q)) - V^l_{c[k]}(\omega^{l,(x)}_{ic[k]}(q)) - \\ IA^l_i(\omega^{l,(x)}_{ic[k]}(q)) - BBE^l_i - (q-1)C_i\end{gathered} \tag{11.29}$$

The maximum busy period for instance q of m_i to meet the local deadline (D_i^l) which we obtained by Eq. (11.26) is inserted into Eq. (11.29), as follows:

$$\begin{aligned} ISB_i^l(q) = \omega_i^l(D_i^l, q) - W_{c[k]}^l(\omega_i^l(D_i^l, q)) - V_{c[k]}^l(\omega_i^l(D_i^l, q)) - \\ IA_i^l(\omega_i^l(D_i^l, q)) - BBE_i^l - (q-1)C_i. \end{aligned} \tag{11.30}$$

As can be perceived, Eq. (11.30) is a function of critical instant combination, local deadline of m_i, and q. For readability, we denote the right side of Eq. (11.30) by the function $Z(\mathcal{I}_c^l[k], D_i^l, q)$ as shown in Eq. (11.31).

$$ISB_i^l(q) = Z(\mathcal{I}_c^l[k], D_i^l, q) \tag{11.31}$$

Then, Eq. (11.28) and Eq. (11.31) are combined as shown in Eq. (11.32).

$$\frac{BW_l}{\alpha_{B,l}^+} * Y(q) = Z(\mathcal{I}_c^l[k], D_i^l, q) \tag{11.32}$$

Consequently, we can derive from Eq. (11.32) the idle slope which makes the instance q of m_i schedulable at the local deadline D_i^l based on instance k of m_c, which is denoted by $\alpha_{B,l}^+(\mathcal{I}_c^l[k], D_i^l, q)$ and is shown in Eq. (11.33).

$$\alpha_{B,l}^+(\mathcal{I}_c^l[k], D_i^l, q) = \left(\frac{Y(q)}{Z(\mathcal{I}_c^l[k], D_i^l, q)}\right) BW_l \tag{11.33}$$

Here, we note that assigning the credit of class B according to the calculations of this section is applicable to the cases where there are multiple class B messages present in the system. When m_i is the only message in class B, there is no same-priority interference, and $Y(q) = 0$. In this case, the credit of class B for transmission of m_i is set according to the utilization of m_i, i.e, $U_i = \frac{C_i}{T_i}$. Finally, the maximum credit out of all calculated credits for each message m_i in class B is set as the minimum credit for the schedulability of all messages in class B, as shown in Eq. (11.34):

$$\alpha_{B,l}^+ = \max_{\substack{\forall i \in \{classB\} \wedge \ l \in \mathcal{L}_i \\ \forall c \in \{classST\} \wedge \ \forall k \in \mathcal{I}_c^l}} \{ \max_{q=1..q_{max}} \{\alpha_{B,l}^+(\mathcal{I}_c^l[k], D_i^l, q)\}\}. \tag{11.34}$$

Algorithm 1 presents the procedure for calculating the minimum credit of class B for the schedulability of the messages on link l. This is an iterative algorithm that updates the credit and q_{max} until the instance of a message in class B that requires the maximum credit is found. Throughout the algorithm, we indicate the previous and the next value of credit and q_{max}, subsequently with the superscript (x) and $(x+1)$. However, the converged values of these parameters are presented without these superscripts.

Algorithm 1 Minimum idle slope of class B on link l.

```
procedure MINCREDITCLASSB($\alpha^{+}_{A,l}$)
    $\alpha^{+,(x)}_{B,l} = U_{B,l}$;
    if $\alpha^{+,(x)}_{B,l} > BW_l - (U_{ST,l} + \alpha^{+}_{A,l})$ then
        $schedulable = False$ ;
        break; //not schedulable.
    end if
    for $m_i$ in $classB$ do
        for $m_c$ in $classST$ do
            $q^{(x)}_{max} = q^{(x+1)}_{max} = 1$;
            $\alpha^{+,(x)}_{B,l}$ = Slope($\mathcal{I}^l_c$, $D^l_i$, $q^{(x)}_{max}$); //Eq. (33).
            $w^l_i$ = BusyPeriod($D^l_i$, $q^{(x)}_{max}$); //Eq. (26).
            $q^{(x+1)}_{max}$ = Qmax($w^l_i$, $\alpha^{+,(x)}_{B,l}$); //Eq. (16).
            if $q^{(x+1)}_{max} > q^{(x)}_{max}$ then
                $q^{(x)}_{max} = q^{(x+1)}_{max}$; //update $q_{max}$.
            end if
            for $q$ in $[2 \ldots q^{(x)}_{max}]$ do
                $\alpha^{+,(x+1)}_{B,l}$ = Slope($\mathcal{I}^l_c$, $D^l_i$, $q$); //Eq. (33).
                if $\alpha^{+,(x+1)}_{B,l} \geq BW_l - (U_{ST,l} + \alpha^{+}_{A,l})$ then
                    $schedulable = False$ ;
                    break; //not schedulable.
                end if
                if $\alpha^{+,(x+1)}_{B,l} > \alpha^{+,(x)}_{B,l}$ then
                    $\alpha^{+,(x)}_{B,l} = \alpha^{+,(x+1)}_{B,l}$; //update $\alpha^{+}_{B,l}$.
                    $w^l_i$ = BusyPeriod($D^l_i$, $q$); //Eq. (26).
                    $q^{(x)}_{max}$ = Qmax($w^l_i$, $\alpha^{+,(x)}_{B,l}$); //Eq. (16).
                    if $q^{(x)}_{max} > q^{(x+1)}_{max}$ then
                        break; //Not maximum $q_{max}$.
                    end if
                    if $q^{(x)}_{max} \leq q^{(x+1)}_{max}$ then
                        $q^{(x)}_{max} = q^{(x+1)}_{max}$; //update $q_{max}$.
                    end if
                end if
            end for
        end for
    end for
    if $\alpha^{+,(x)}_{B,l} > \alpha^{+}_{B,l}$ then
        $\alpha^{+}_{B,l} = \alpha^{+,(x)}_{B,l}$;
    end if
    return $\alpha^{+}_{B,l}$
end procedure
```

We assume that the credit of class A is calculated via Eq. (11.22) and it is input to the procedure in Algorithm 1. On line 2 of Algorithm 1, the starting value for credit of class B on link l is set to the utilization of this class. Further, from line 3 to line 6, the algorithm checks whether there is enough bandwidth available for transmission of class B on link l.

The loop starting from line 7 to line 35 iterates through all the messages in class B to find the instance of a message that requires the maximum credit, since this credit is the minimum safe credit to make class B schedulable. Besides, the loop starting from line 8 to line 34 iterates through all critical instant candidates in class ST to account for the preemption and overhead by ST traffic in the maximum credit of a message. On line 9, q_{max} is set to one because we first analyze the credit for only one instance of the message under analysis. On line 10, the credit of class B is set according to the proposed analysis in Eq. (11.33). Then, the maximum busy period is calculated via Eq. (11.26). The new q_{max} is found using Eq. (11.16) and based on the credit and maximum busy period which were calculated on lines 10 and line 11. Subsequently, on line 13 to line 15, we determine whether q_{max} needs to be updated to a value more than 1. The next loop starting from line 16 to line 33 iterates in case q_{max} is more than 1 for analyzing the maximum credit in all q_{max} number of instances of the message under analysis.

On line 17 the credit for instance q of m_i is calculated. The procedure must stop, in case this credit is more than the bandwidth left after transmission of the traffic in class ST and class A. Because this instance makes class B unschedulable. We check the availability of bandwidth based on the last update of credit for instance q of m_i on lines 18 to 21.

If instance q of m_i is schedulable and the credit in this iteration is more than the credit in the previous iteration, we update the credit on line 23. Then, the procedure continues to calculate the busy period for this instance (line 24). Accordingly, q_{max} is updated on line 25 based on the current value of credit. On lines 26 to 28, we check if q_{max} is not increasing. In this case, we do not need to check any larger q, thus we break the loop. However, on lines 29 to 31, we check if q_{max} has increased in this iteration, then q_{max} is updated for increasing the number of instances to count in the analysis (line 30). Finally, on lines 36 to 38, the maximum credit among all messages is set as the safe credit for making all messages in class B schedulable.

11.5.3 Local deadline assignment

The local deadline is user-defined and can be set with different considerations such as load of traffic per link. However, there are various methods to define the local deadlines [11, 2]. In this paper, we use the same method used in [2] to define local deadlines per link in the route of the message m_i. This method relies on calculating the relation between the load of m_i on one link in its path and the total load on all the links in its route (including all the other traffic in the system). This relation is specified for each link by the coefficient (K_i^l) as shown in Eq. (11.35).

$$K_i^l = \frac{load_i^l}{\sum\limits_{L=1..|L_i|} load_i^L} \tag{11.35}$$

where, $load_i^l$ is calculated using Eq. (11.36):

$$load_i^l = U_{i,l} + \max_{m_j \in lp(m_i)} \left\{ U_{j,l} \right\} + \sum_{\substack{m_j \in (hp(m_i) \vee \\ sp(m_i))}} \left\{ U_{j,l} \right\} + \sum_{m_j \in \{classST\}} \left\{ U_{j,l} \right\}. \tag{11.36}$$

The link load coefficient is used to calculate the local deadline on each link as a proportion of the deadline of the message as shown in Eq. (11.37).

$$D_i^l = K_i^l.D_i \tag{11.37}$$

11.6 Assessment of the proposed solution

This section presents an evaluation of an industrial vehicular use case to show the impact of using the bandwidth reservation solution compared to the recommendation by the standards.

11.6.1 Use-case scenario

In this section, we present a use-case scenario that is used for the evaluation of the proposed BRA. Figure 11.3 shows the vehicular application use case which is inspired by [12]. In this use case, there are 8 end-stations and two TSN switches. The total bandwidth is $100Mbps$ and the switch fabrication delay in both switches is assumed to be $5\mu s$. Table 11.1 shows the set of messages in the use case. There are in total fifteen messages which include three messages from class ST, four messages from class A, five messages from class B, and three messages from class BE. Two of the end-stations act as only receivers, namely Head Unit, and AVSink. All the other end-stations send messages to the Head Unit and AVSink.

11.6.2 Evaluations

In the set of evaluations, we calculate the minimum required bandwidth for the set of AVB traffic in order to ensure the schedulability of the traffic in each AVB class. Table

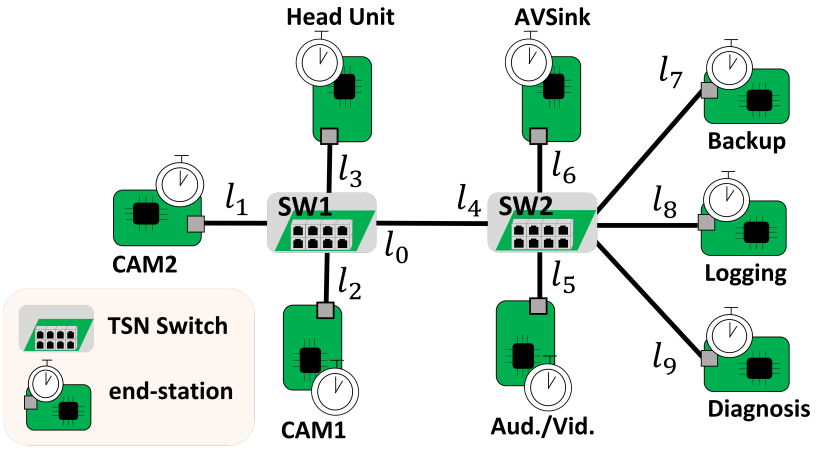


Figure 11.3. Evaluation use case inspired from [12].

Table 11.1. Traffic in the use case (ms).

m_i	Sender	Receiver	$T_i = D_i$	C_i	P_i
m_0	CAM1	Head unit	200	0.00496	ST
m_1	CAM1	Head unit	200	0.00496	ST
m_2	CAM2	Head unit	200	0.00496	ST
m_3	CAM1	Head unit	100	0.06624	A
m_4	CAM1	Head unit	50	0.06624	A
m_5	CAM2	Head unit	100	0.06624	A
m_6	CAM2	Head unit	50	0.06624	A
m_7	CAM1	Head unit	100	0.06624	B
m_8	CAM1	Head unit	50	0.12112	B
m_9	CAM2	Head unit	100	0.06624	B
m_{10}	CAM2	Head unit	50	0.06624	B
m_{11}	Aud./Vid.	AVSink	100	0.06624	B
m_{12}	Backup	Head unit	100	0.04336	BE
m_{13}	Logging	Head unit	110	0.12336	BE
m_{14}	Diagnosis	Head unit	120	0.16672	BE

11.2 presents the credit allocation based on the utilization of classes A and B on the links. Furthermore, Table 11.3 presents the set of results for credit allocation by the proposed BRA.

In this evaluation, there are various traffic on each of the links in the topology. More specifically, l_1 is connected to CAM2 which generates messages m_0 and m_1 from class ST, m_3 and m_4 from class A, and m_7 and m_8 from B. l_2 is connected to CAM1 which generates messages m_2 from class ST, m_5 and m_6 from class A, and m_9 and m_{10} from B. l_3 is used by all the messages except for m_{11}, i.e., all traffic classes cross the link l_3. Link l_4 is used by message m_{11} from class B, and the messages m_{12}, m_{13} and m_{14} from class BE.

As can be seen in Tables 11.2 and 11.3, the bandwidth reserved for class A and B on the links l_1, l_2, and l_3 based on the utilization-based bandwidth reservation (i.e., according to the standard recommendations) is significantly less than the bandwidth calculated by the proposed analysis. In this example, the under-reservation of bandwidth in Table 11.2 leads to unschedulable network. For instance, in order for messages in class A to meet their local deadlines on l_2, m_3 it requires 0.222 $Mbps$ bandwidth, and m_4 requires 0.450 $Mbps$ bandwidth (i.e., the local deadline on l_2 is 29.79 ms for m_3 and 14.89 ms for m_4). Besides, according to the proposed analysis, message m_7 requires 0.409 $Mbps$, and m_8 requires 0.452 $Mbps$ on l_2 to meet their local deadlines (i.e., the local deadline on l_2 is 29.79 ms for m_7 and 14.89 ms for m_8). As calculated by the proposed BRA in Table 11.3, the minimum reserved bandwidth on l_2 for classes A and B is set according to the required bandwidth of m_4 and m_8 respectively. Because these messages demand the highest bandwidth of their classes. It is apparent that messages m_4 and m_8 cannot gain sufficient bandwidth and miss their deadlines if the bandwidth is set according to the recommended calculations by the standards, as shown in Table 11.2.

The utilization-based bandwidth reservation is more applicable than the proposed BRA in some special cases. For example, the required bandwidth for m_{11} on l_4 is set based on its utilization since there are no other same-priority messages on l_4. Similarly, message m_{11} from class B is the only message that uses the links l_5 and l_6. Therefore, the bandwidth reserved for the links l_5 and l_6 are similar based on the utilization-based and the proposed bandwidth reservation approaches.

Table 11.2. utilization-based bandwidth reservation.

$Mbps$	l_1	l_2	l_3	l_4	l_5	l_6
$\alpha^+_{A,l} * R$	0.198	0.198	0.397	NA	NA	NA
$\alpha^+_{B,l} * R$	0.198	0.308	0.507	0.066	0.066	0.066

Table 11.3. Bandwidth reserved by the proposed BRA.

$Mbps$	l_1	l_2	l_3	l_4	l_5	l_6
$\alpha^+_{A,l} * R$	0.538	0.450	0.993	NA	NA	NA
$\alpha^+_{B,l} * R$	0.541	0.452	1.030	0.066	0.066	0.066

11.7 Conclusion and future work

In this paper, we proposed a Bandwidth Reservation Analysis (BRA) for Audio-Video Bridging (AVB) traffic in Time-Sensitive Networking (TSN), which is based on the worst-case Response Time Analysis (RTA). The analysis calculates the minimum bandwidth that is required for traffic classes A and B to become schedulable. We show that the standard recommendations for the bandwidth allocation of AVB traffic classes are not sufficient and in many cases lead to unscheduealble system. In our proposed BRA, we consider the TSN network with AVB traffic classes and ST traffic when ST can preempt other lower-priority classes. This solution allows the TSN network designers to allocate credits efficiently for their AVB traffic classes ensuring the schedulability of their system. The solution is agnostic of the ST offline schedule, which can be done before this analysis with any scheduling solution that exists in the literature. Note that the solution is pseudo-polynomial with respect to the number of messages and the network size, which makes it suitable in practice. We showed the impact of the BRA on an industrial vehicular use case.

Acknowledgments

The work presented in this paper was supported by the Swedish Governmental Agency for Innovation Systems (VINNOVA) via the DESTINE, PROVIDENT and INTER-CONNECT projects and the Swedish Knowledge Foundation via the DPAC and HERO projects.

Bibliography

[1] *802.1Q-2022 - IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks*. IEEE, 2022.

[2] Mohammad Ashjaei, Gaetano Patti, Moris Behnam, Thomas Nolte, Giuliana Alderisi, and Lucia Lo Bello. Schedulability Analysis of Ethernet Audio Video Bridging Networks with Scheduled Traffic Support. *Real-Time Systems*, 2017.

[3] Jingyue Cao, Mohammad Ashjaei, Pieter J. L. Cuijpers, Reinder J. Bril, and Johan J. Lukkien. An Independent yet Efficient Analysis of Bandwidth Reservation for Credit-based Shaping. In *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*, 2018.

[4] Giuliana Alderisi, Gaetano Patti, and Lucia Lo Bello. Introducing Support for Scheduled Traffic over IEEE Audio Video Bridging Networks. In *18th IEEE Conference on Emerging Technologies Factory Automation*, 2013.

[5] Reinder J Bril, Hamid Hassani, Pieter JL Cuijpers, and Geoffrey Nelissen. Cost of Robustness of Independent WCRT Analysis for CBS of Ethernet AVB Using Eligible Intervals. In *2023 IEEE 19th International Conference on Factory Communication Systems (WFCS)*. IEEE, 2023.

[6] L. Lo Bello, M. Ashjaei, G. Patti, and M. Behnam. Schedulability Analysis of Time-Sensitive Networks with Scheduled Traffic and Preemption Support. *Journal of Parallel and Distributed Computing*, 2020.

[7] Ershuai Li, Feng He, Qiao Li, and Huagang Xiong. Bandwidth Allocation of Stream-reservation Traffic in TSN. *IEEE Transactions on Network and Service Management*, 2021.

[8] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, S. Afshar, and S. Mubeen. Schedulability Analysis of Best-effort Traffic in TSN Networks. In *26th IEEE*

International Conference on Emerging Technologies and Factory Automation, 2021.

[9] Thomas Stüber, Lukas Osswald, Steffen Lindner, and Michael Menth. A Survey of Scheduling in Time-Sensitive Networking (TSN). *arXiv preprint arXiv:2211.10954*, 2022.

[10] J. Maki-Turja and M. Nolin. Fast and Tight Response-Times for Tasks with Offsets. In *Euromicro Conference on Real-Time Systems*, 2005.

[11] Juan M. Rivas, J. Javier Gutierrez, J. Carlos Palencia, and Michael Gonzalez Harbour. Deadline Assignment in EDF Schedulers for Real-Time Distributed Systems. *IEEE transactions on parallel and distributed systems*, 2015.

[12] H. Lim, K. Weckemann, and D. Herrscher. Performance Study of an In-car Switched Ethernet Network without Prioritization. In *Communication Technologies for Vehicles*, 2011.