TECHNISCHE UNIVERSITÄT MÜNCHEN
Lehrstuhl für Robotik, Künstliche Intelligenz und Echtzeitsysteme

A Modeling Framework to Facilitate Schedule Synthesis of Time-Sensitive Networking

Morteza Hashemi Farzaneh

Vollständiger Abdruck der von der Fakultät der Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr.-Ing Georg Carle
Prüfer der Dissertation:	1. Prof. Dr.-Ing. habil. Alois Christian Knoll
	2. Prof. Miroslav Pajic, Ph.D.

Die Dissertation wurde am 09.10.2018 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 29.03.2019 angenommen.

Abstract

Over the last two decades, various proprietary modifications have been applied to Ethernet technology to fill the gap of supporting determinism required for time-critical systems. Time-Sensitive Networking (TSN) standards are under development by the Institute of Electrical and Electronic Engineers (IEEE) and address hard timing requirements of Ethernet-based distributed applications. The main objective of these standards is to support distributed applications with different levels of criticality in the same networking infrastructure. The support of mixed-criticality reduces heterogeneity of networking technologies and simplifies not only the development but also the integration of distributed applications.

However, the traffic planning and verification of these networks require advanced expertise and are time-consuming.

In this thesis, a modeling approach is presented based on logic programing to facilitate network scheduling and infrastructural verification and to enable the analysis of design errors. A graphical editor is conceived and implemented containing modeling elements (topology, data flows, and requirements) to create adequate network models. These models are automatically translated into a network knowledge base which is the core of the framework. It is used to process queries for verification and selective information gathering in order to build scheduling constraints. Based on a constraint-labeling approach, design errors in model instances are retraced and iteratively corrected. Moreover, an approach is proposed based on the data-centricity paradigm to enable network autoconfiguration at run-time.

The feasibility of the approach is evaluated using exemplary networking scenarios and an experimental setup. The demonstrations emphasize that significant parts of the manual TSN configuration and verification can be automated using the developed framework. Hence, this thesis contributes to simplification and acceleration of implementing TSN-based applications.

Zusammenfassung

In den letzten zwei Jahrzehnten wurden verschiedene proprietäre Modifikationen auf die Ethernet-Technologie angewendet. Das Ziel dieser Modifikationen war, den notwendigen Determinismus zu ermöglichen, der von zeitkritischen Systemen angefordert wird. Time-Sensitive Networking (TSN) Standards, die vom Institute of Electrical and Electronic Engineers (IEEE) entwickelt werden, adressieren harte Echtzeitanforderungen der Ethernet-basierten verteilten Anwendungen. Das Hauptziel dieser Standards besteht darin, die Implementierung verteilter Anwendungen mit unterschiedlichen Kritikalitätsstufen in derselben Netzwerkinfrastruktur zu unterstützen. Die Unterstützung von Mixed-Criticality reduziert die Heterogenität der Vernetzungstechnologien und vereinfacht die Entwicklung und Integration verteilter Anwendungen.

Die Planung und Verifikation dieser Netzwerke sind zeitintensiv und verlangen fortgeschrittene Fachkenntnisse.

In dieser Arbeit wird daher ein Modellierungsansatz basierend auf logischer Programmierung entwickelt, um die Netzwerkplanung und infrastrukturelle Verifikation zu unterstützen und die Analyse der Entwurfsfehler zu ermöglichen. Zur Erstellung von adäquaten Netzwerkmodellen wird ein grafischer Editor mit entsprechenden Modellierungselementen (Topologie, Datenflüsse und Anforderungen) konzipiert und implementiert. Die erstellten Modelle werden automatisch in eine Netzwerk-Wissensbasis übersetzt, die den Kern des Frameworks bildet. Sie ermöglicht Abfragen zur Verifizierung und selektiven Informationserfassung (z.B. notwendig für die Generierung der Planungsrandbedingungen). Basierend auf einem Randbedingungskennzeichnungsansatz werden Entwurfsfehler in Modellinstanzen zurückverfolgt und iterativ korrigiert. Darüber hinaus wird ein Ansatz entwickelt, der das Konzept der datenzentrischen Kommunikation nutzt, um die Autokonfiguration zur Laufzeit zu ermöglichen.

Die Umsetzbarkeit des Ansatzes wird mit Hilfe beispielhafter Vernetzungszenarien und eines experimentellen Aufbaus evaluiert. Diese Beispiele verdeutlichen, dass signifikante Teile der manuellen Konfiguration und Verifikation der TSN-Netzwerke mit Hilfe des entwickelten Frameworks automatisiert werden können. Deshalb trägt diese Arbeit zur Vereinfachung und Beschleunigung der Entwicklung TSN-basierter Anwendungen bei.

Acknowledgements

I would like to express my sincere gratitude to Professor Alois Knoll for giving me the opportunity to join his esteemed research group. He has always supported me with fruitful discussions, motivating feedback, and future-oriented suggestions.

I would like to thank all my colleagues at the Chair of Robotics, Artificial Intelligence, and Real-time Systems for their fantastic co-operation. I want to thank Amy Bücherl for her continues support and proof reading of the thesis. I also want to thank Ute Lomp for permanent support and helping me with the organization of the dissertation process. I want to thank Marie-Luise for her support in my project management tasks at the chair. I would like to thank Dr. Suraj Nair, Dr. Ali Nasseri, Dr. Alejandro Mendoza, Brian Jensen, Philipp Heise, Dr. Daniel Renjewski, Dr. Alexander Lenz, Sina Shafaei, Dr. Claus Lenz, Dr. Hardik Shah, Professor Matthias Althoff, Dr. Kai Huang, and Dr. Stefan Kugele for the productive discussions.

My special thanks to my wife, Helena, for her love and proof reading of the thesis. Many thanks to my lovely children Hossein and Minou who provide fun in my life.

I want to express my special thanks to my parents, Effat and Javad, and my sister, Atefe, who have always supported me with their love and prayers from long distances. This work is dedicated to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Standard switched Ethernet is a wide-spread, successful, low-cost, high-bandwidth and future-oriented infrastructure that is used in home and office networks. Due to the lack of real-time capabilities, it does not fulfill hard timing requirements. The main issue is the best effort strategy of the Ethernet which leads to the nondeterministic transmission of competing Ethernet frames.

Based on the advantages of Ethernet and interests of the factory automation domain, it has been modified to enable deterministic and real-time communication. The result is a set of Ethernet-based technologies such as e. g. Time-Triggered Ethernet [1, 2] and EtherCAT [3] which provide timing guarantees. Almost all of them require significant modifications in hardware, network, and application stack to achieve real-time guarantees. Ethernet Powerlink [4] for instance uses obsolete network hubs to achieve the lowest latency and does not use the capabilities of switched Ethernet in the duplex mode which can be considered a waste of bandwidth. EtherCAT slave stack is implemented in Field Programmable Gate Array (FPGA) to achieve the highest performance and minimum cycle times. Moreover, the communication control is part of the protocol and tightly coupled with the proprietary application layer. In our previous work, a subset of known Industrial Ethernet technologies is analyzed and compared [5].

Based on the timing requirements of the audio and video transmission application domain, the Institute of Electrical and Electronics Engineers (IEEE) decided to extend Ethernet standards to develop real-time Audio Video Bridging (AVB) [6] technologies. AVB is an IEEE working group which defines a set of standards to achieve timing and synchronization guaran-

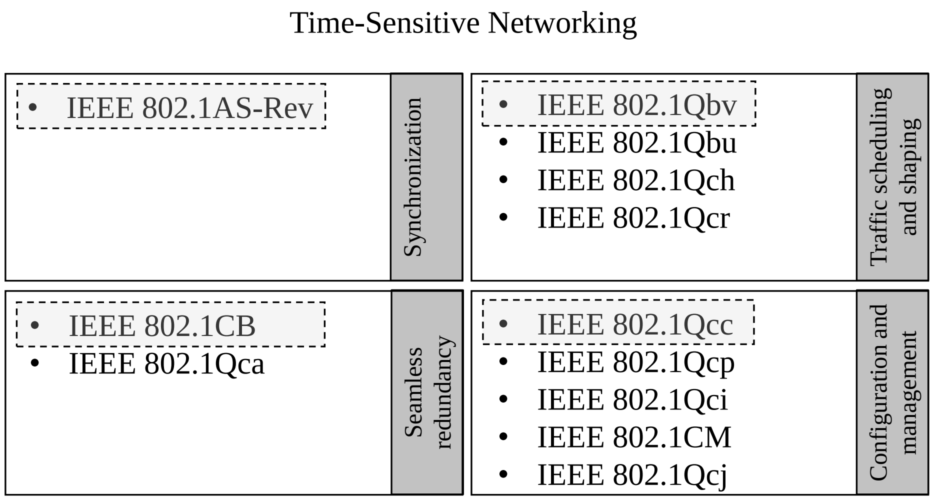


Figure 1.1: Overview of standardization activities in TSN work group. Highlighted standards (drafts) are relevant in the context of this thesis.

tees. Its objective is to guarantee a worst-case latency of $2ms$ for the high-priority traffic class A and worst-case latency $50ms$ for traffic Class B over a maximum of seven hops. To guarantee timing, Credit-Based Shaper (CBS) [7], Stream Reservation Protocol (SRP) [8] and timing and synchronization [9] have been specified in IEEE 802.1Qav, IEEE 802.1Qat, and IEEE 802.1AS. Applying CBS, network bursts (transmission of multiple frames directly behind each other) are controlled such that reserved AVB streams fulfill their timing requirements.

The motivation for further developments and enhancements of AVB standards is originated in the hard timing and safety requirements of factory automation and automotive applications. The main motivation is to reduce the worst-case latency and jitter down to microsecond range. It also aims to develop important additional features such as fault-tolerance mechanisms to meet safety-relevant requirements such as fail-operational. The Time-Sensitive Networking (TSN) [10] task group aims at developing the required standards towards a unified hard real-time Ethernet technology.

1.1.1 Time-Sensitive Networking

The standards on which the TSN task group is working can be divided into four categories as depicted in Figure 1.1. A short summary of the standards is presented.

Synchronization: Time synchronization plays a significant role in real-time communication and means that all network devices involved in a real-time application require the same time reference. To achieve this, the device clocks need to be synchronized. The TSN synchronization specification which is currently under development is based on the IEEE 1588 standard. The new version IEEE 802.1AS-Rev aims to incorporate the accepted features of one-step transparent clocks [11]. The main improvement is to consider error situations and to handle them (e. g. to find a solution in case of failure of a physical communication link or a master clock). Moreover, the new version is designed to deal with different time domains in switches and end-stations [12].

Traffic scheduling and shaping: IEEE 802.1 Qbv Time-Aware Shaper (TAS) proposes a time-triggered scheduling approach to prioritize and control frame forwarding at egress ports of the switches and end-stations [13] (in the context of this thesis, both terms *frame* and *stream* are used to describe a dataflow between a sender and receiver. We use *stream* when the periodic dataflow could contain more than one Ethernet *frame*. We use the term *frame*, when the concepts of TSN standards are explained which are applied on single Ethernet frames). Its main objective is to minimize the latency and jitter of high-priority frames. Virtual Local Area Network (VLAN) tag is used to encode priority values from 0 to 8 which are inserted into frames allowing support of mixed-critical streams over Local Area Networks (LANs). TAS has to ensure that the Ethernet interface is not transmitting any low-priority frame on an egress port when a time-triggered frame has to be sent. To guarantee this, a guard band is reserved before the time slot of a time-triggered frame. Within the guard band, the start of transmission for all frames is blocked. The length of a guard band window is equal to the maximum frame size which may be transmitted.

The time slots which are occupied by guard bands reduce the bandwidth utilization for low-priority traffic. To reduce these negative effects, a frame preemption mechanism is specified in IEEE 802.1Qbu [14]. The main idea is to preempt a long low-priority frame in favor of a high-priority time-triggered frame. The capability of frame preemption reduces the size of guard band windows and therefore increases the bandwidth utilization.

Another ongoing standard which deals with traffic shaping is IEEE 802.1Qch [15] which also deals with scheduling. It aims to reduce the configuration complexity by relaxing the timing requirements. The main concept is to supports a cyclic queuing and forwarding mechanism such that the received frames within a cycle are collected and transmitted to the next hop which will happen in the next consecutive cycle. Hence, the worst-case latency can be calculated using the number of hops. This standard also requires network synchronization similar to the time-aware shaper. In contrast, a recently approved draft IEEE 802.1Qcr [16] deals with asynchronous traffic shaping which does not require synchronization and aims to increase the bandwidth utilization for low-priority frames.

Seamless redundancy: To support fail-operational and seamless redundancy requirements (common in many critical distributed applications), IEEE 802.1CB standard is under development to support frame replication and elimination [17]. The approach is similar to High-Availability Seamless Redundancy (HSR) and the Parallel Redundancy Protocol (PRP). The frames are duplicated and transmitted in parallel via disjoint network paths. To avoid the misunderstanding of multiple replicated frames at the receiver end-stations, the frame duplicates have to be detected and eliminated at the receiver side. To support seamless redundancy, it is crucial to find disjoint redundant paths by obtaining information about the network topology. Topology discovery and bandwidth reservation are treated in IEEE 802.1Qca standard [18, 19].

Management and configuration: The configuration and management of TSN networks is the topic of IEEE 802.1Qcc standard [20] in which different configuration approaches are proposed. The centralized approach assumes a central entity in the network which has an overview of all network components and is responsible for configuration decisions based on the requirements. In contrast, in the decentralized approach, each network component uses its locally available information for configuration activities.

In IEEE 802.1Qcp, a Unified Modeling Language (UML) based information model and a YANG data model are specified. These models aim to support configuration and status reporting for bridges [21]. The draft of IEEE 802.1Qcj [22] draft intends to specify protocol and procedures enabling the auto-attachment of network devices to provider backbone service instances. To ensure that all TSN nodes' behavior conforms to the configuration, IEEE 802.1Qci [23] standard is being developed. The main objective is to specify per-stream filtering and policing mechanisms which detect and mitigate stream transmissions which show troubling behavior. This is also highly related to security aspects of the network when a malicious (e. g. hijacked) end-station produces more traffic than reserved. Such misbehaviors prevent other streams to fulfill their requirements. Finally, IEEE 802.1CM [24] deals with the transmission of time-sensitive fronthaul streams in TSN networks.

In the context of this thesis, we discuss the significant role of node synchronization (IEEE 802.1AS-Rev based on IEEE 1588 v2 PTP) as an infrastructural requirement for time-triggered communication. We investigate how the configuration of IEEE 802.1Qbv nodes can be automated. The developed concepts show how redundancy requirements which are e. g. related to

IEEE 802.1CB can be verified. Moreover, we explain the relation of the developed modeling framework in this thesis to the configuration draft IEEE 802.1Qcc. The remaining standards **are out of the scope** of this thesis.

The main focus of this thesis is on the configuration and verification of the TSN mechanisms which enable standardized time-triggered communication in switched Ethernet networks. Thus, the corresponding standard IEEE 802.1Qbv is explained in detail.

1.1.2 Time-Aware Shaper IEEE 802.1Qbv

Fulfilling tight timing requirements of time-critical traffic is not possible if only packet prioritizing mechanism based on the First-In-First-Out (FIFO) principle is applied (cf. *IEEE 802.1Q* and *IEEE 802.1P*). If a frame with lower priority is going to be transmitted on an egress port, it has to be guaranteed that its transmission will be completed before a frame with a higher priority arrives for transmission. Without this guarantee, in a worst-case scenario, a high-priority frame may experience a delay of a maximum-sized Ethernet frame based on the defined Maximum Transmission Unit (MTU). The sum of such delays over multiple hops increases the transmission latency and could result in missing end-to-end deadlines. Moreover, the end-to-end transmission jitter becomes non-deterministic, because the delays at each hop are nondeterministic. The most significant contribution of TSN to the standard Ethernet is the standardization of the time-triggered communication paradigm (TAS) specified in IEEE 802.1Qbv (regarding enhancements for scheduled traffic). TAS supports applications which require highly predictable frame delivery with lowest latency and jitter. Such applications are common in e. g. the industrial automation and automotive domain, for example, where critical data is transmitted periodically using a pre-defined static schedule. Missing deadlines of such critical frames can result in inaccuracy or system failure. Using TAS mechanism, it is possible to mix the time-critical traffic with less time-critical traffic in the same network.

The core idea is that at each arbitrary point in time, TAS can control, which traffic priority class or classes have exclusive link access to transmit data. To avoid that low-priority traffic disturbs the transmission of time-critical traffic, the transmission of low-priority traffic has to be blocked in advance. This time slot is called guard-band, which starts before the transmission of the time-critical traffic. Exemplary scheduling using TAS is demonstrated in Figure 1.2. At a given egress port, nine streams are competing to get the link access. Streams S_1, S_2, and S_3 are time-critical with periods of $10ms$, $10ms$, and $20ms$ and priority class 7. The non-critical

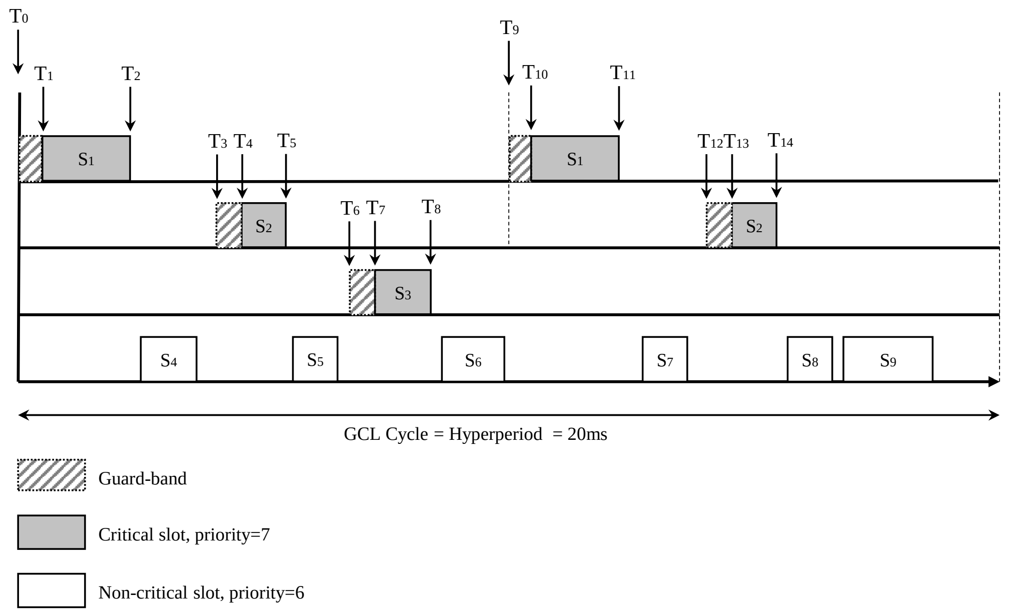

T0
T1
T2
S1
T3 T4
T5
S2
T6 T7
T8
S3
T9
T10
T11
S1
T12T13
T14
S2
S4
S5
S6
S7
S8
S9
GCL Cycle = Hyperperiod = 20ms
Guard-band
Critical slot, priority=7
Non-critical slot, priority=6

Figure 1.2: Mixed-criticality with Time-Aware Shaper (IEEE 802.1Qbv). The conflict-free transmission of periodic time-critical traffic is guaranteed.

streams S_4, S_5, S_6, S_7, S_8, S_9 have priority 6 and are not transmitted within the defined guard-bands.

The reserved slots for transmission of these streams do not overlap with other priority classes (as specified in IEEE 802.1Qbv to minimize buffering of time-critical streams). Within guard-bands, no frames can be transmitted. The remaining time is available for transmission of other priority classes.

Each Ethernet port has eight time-aware gates for eight priority classes from 7 (highest) to 0. A gate driver opens and closes these gates based on a predefined schedule stored in Gate Control List (GCL). To use the previously presented sample schedule, the schedule has to be transformed as GCL entries. As presented in Figure 1.3, at each point in time, gates can be either closed (0) or opened (1). At T_0 there is a guard-band and therefore no frames can be transmitted and all gates are closed. At T_1, T_4, T_7, T_{10}, and T_{13} the gates are only open for priority class 7 which time-critical streams S_1, S_2, and S_3 are scheduled (cf. Figure 1.2). When the end of a critical slot arrives e. g. at T_2, other gates are opened for other priority classes such as priority class 6 in which the remaining streams get access to the link. The length of the GCL cycle depends on the hyper period of all competing time-critical streams. Each queue can apply its own transmission selection algorithm. It can be a simple FIFO mechanism or a

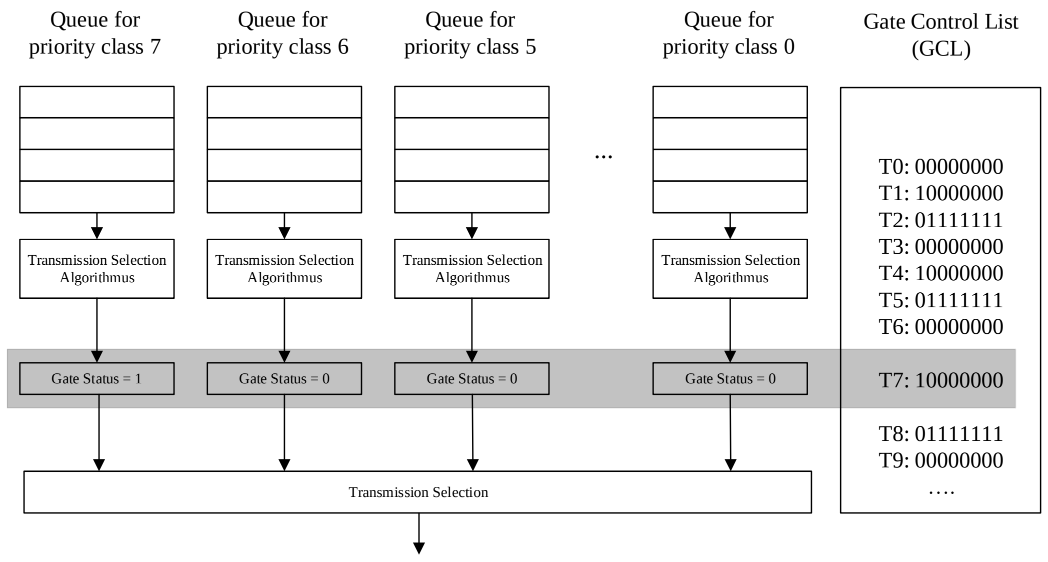


Figure 1.3: An exemplary Gate Control List (GCL)

credit-based shaper as specified in [7]. When a new GCL cycle starts, the opening and closing events will be repeated.

1.2 Motivation and Research Questions

TSN standards have the potential to build an open, vendor-independent basis technology for real-time communication. The support of mixed-criticality (co-existence of applications with different levels of criticality in the same networking) reduces the heterogeneity of networking technologies and simplifies not only the development but also the integration of distributed applications.

However, the traffic planning and verification of these networks require advanced expertise and are time-consuming. Assisting tools can mitigate the configuration and verification challenges for network designers and application developers. Following this motivation, we aim to answer three research questions in the context of automated configuration and verification of TSN networks.

RQ1*: How can the complexity of TSN schedule synthesis be reduced?*

To justify this question, first the term *complexity* has to be explained in the context of this work. Finding a feasible time-triggered schedule is a known NP-complete problem. Hence, there is no available efficient algorithm which can be applied to find a feasible schedule. To tackle this problem, computer-based constraint solving approaches are applied. In the literature, schedule synthesis is used as a term to describe this problem and approaches to deal with it (cf. Chapter 2). All necessary steps required for extracting relevant information for scheduling, understanding them, transforming them into mathematical constraints, and distributing them among the network nodes build together a complex heterogeneous task list.

Considering the information available about a TSN network, we define two different views. The network topology view focuses on the information related to the physical components e. g. switches, end-stations, cables, and their capabilities. For instance, an end-station either supports synchronization and IEEE 802.1Qbv or does not support this standard. In the dataflow view, the focus is on the applications and their timing or redundancy requirements. For example, an end-station can send a stream with a certain timing requirement to a receiving end-station (a communication which is required to implement a time-critical application). Obtaining the network information and extracting the relevant parts for schedule synthesis, requires an efficient approach where the required information can be queried with a minimum of implementation overhead (software routine) for each new type of query.

The extracted information has to be translated into scheduling constraints which are then solved by constraint solvers. From the perspective of network designers and end-users, details such as how the information is obtained, how constraints are built, and which solvers are used to find a feasible schedule, are less relevant. Obviously, it is more worthwhile to have a possibility to model and design TSN networks in order to automate the configuration procedures hiding the unnecessary details and complexities. To achieve this, the schedules computed by solvers have to be prepared and distributed among switches and end-stations.

RQ2: *How to facilitate model verification before schedule synthesis and support infeasibility analysis after synthesis?*

Designed network models are not always error-free. A design error is defined as a logical mistake by the designer. For instance, a time-triggered stream can be modeled with tightest timing requirements. Such a stream must travel through network nodes which support IEEE 802.1Qbv and synchronization. A design error can occur when one of the physical nodes in the path does not support synchronization. Ignoring this error in the model can lead to a feasible schedule (calculated by the solver). However, after the distribution of this schedule in the real network, the latency measurement of the stream will show that it misses its timing requirements. A network model has to be verified prior to the schedule synthesis.

Even if all design errors of the model are corrected, there is no guarantee that the set of modeled streams is feasible. In other words, the solver can conclude that there is no feasible schedule for the streams defined in the network model. It is significantly helpful for the designers to be able to localize and backtrace the origin of the infeasibility in order to find a solution.

RQ3*: How to facilitate autoconfigurtion of TSN networks with focus on IEEE 802.1Qbv standard?*

One of the aspects that determine the potential of the TSN standards is the possibility of dynamical automated configuration and verification. Automated network configuration at run-time without involving human expertise, reduces network downtime and managing costs caused by interruption of running applications. This brings us to the last research question of this thesis:

1.3 Contribution

In order to answer the research question **RQ1**, the first contribution fills the identified gap of a graphical modeling tool for TSN which simplify network modeling towards automated schedule synthesis for IEEE 802.1Qbv. A modeling framework is presented. First, a formal meta-model is developed which builds the basis of a network knowledge base containing network details in the form of facts. The logic programming paradigm is used to implement the knowledge base. Secondly, to hide the complexity of logic programming for the network designer, we develop an object-oriented graphical modeling tool. Using the graphical elements of this tool, TSN network architectures can be modeled. A model translator is implemented that use the graphical model and generates network facts for the knowledge base. Thirdly, using the generated facts, the schedule constraints are built and solved to create the necessary configuration parameters for switches and end-stations. We define infrastructural constraints which have to be fulfilled before starting a synthesis.

Regarding the second research question **RQ2**, the second contribution extends the state of the art in time-triggered schedule synthesis by considering and labeling the constraints of IEEE 802.1Qbv (required for schedule synthesis) while they are generated from the model. We present inference rules which can be used to verify the model prior to the synthesis. After the synthesis, if streams are infeasible, we enable the designers to backtrace the origin of the infeasibility.

The labels contain information about the involved hardware components, application domains, stream period, stream size, etc. Using these labels, the designers can localize the problem. The presented approach exploits the newest developed algorithms to find minimal unsatisfiable cores. Using a complex network model as a case study, the approach is evaluated.

The third contribution presents a novel approach to enable IEEE 802.1Qbv for automated network reconfiguration at run-time. A Plug&Configure concept for TSN is proposed to address the third research question **RQ3**. The core idea is to combine the strengths of the data-centricity paradigm with the advantages of TSN (especially synchronization and scheduled traffic). Using data-centric middleware approaches, highly flexible networking based on a publisher-subscriber model is possible. The presented approach proposes that a new device intending to join a running TSN network, has to send its profile to configuration entity in the network. The configuration entity specifies the streams which will be published or consumed based on device profiles. The configuration entity adds the new profile to the existing network knowledge base. If required, a re-synthesis request will be sent to the synthesis entity in the network. The end-station is informed about the status of synthesis and if it is successful, it can start to execute the data-centric application. A TSN experimental setup is built to demonstrate the capabilities of the developed concepts.

Parts of this thesis are published at several international conferences. The most relevant publications are listed below.

- Hashemi Farzaneh, Morteza; Kugele, Stefan; Knoll, Alois: A Graphical Modeling Tool Supporting Automated Schedule Synthesis for Time-Sensitive Networking, in: IEEE International Conference on Emerging Technologies And Factory Automation (ETFA), 2017

- Hashemi Farzaneh, Morteza; Knoll, Alois: Time-Sensitive Networking (TSN): An Experimental Setup, in: IEEE Vehicular Networking Conference (VNC), 2017

- Hashemi Farzaneh, Morteza; Shafaei, Sina; Knoll, Alois: Formally verifiable modeling of in-vehicle time-sensitive networks (TSN) based on logic programming, in: IEEE Vehicular Networking Conference (VNC), 2016

- Hashemi Farzaneh, Morteza; Knoll, Alois: An ontology-based Plug-and-Play approach for in-vehicle Time-Sensitive Networking (TSN), in: IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016

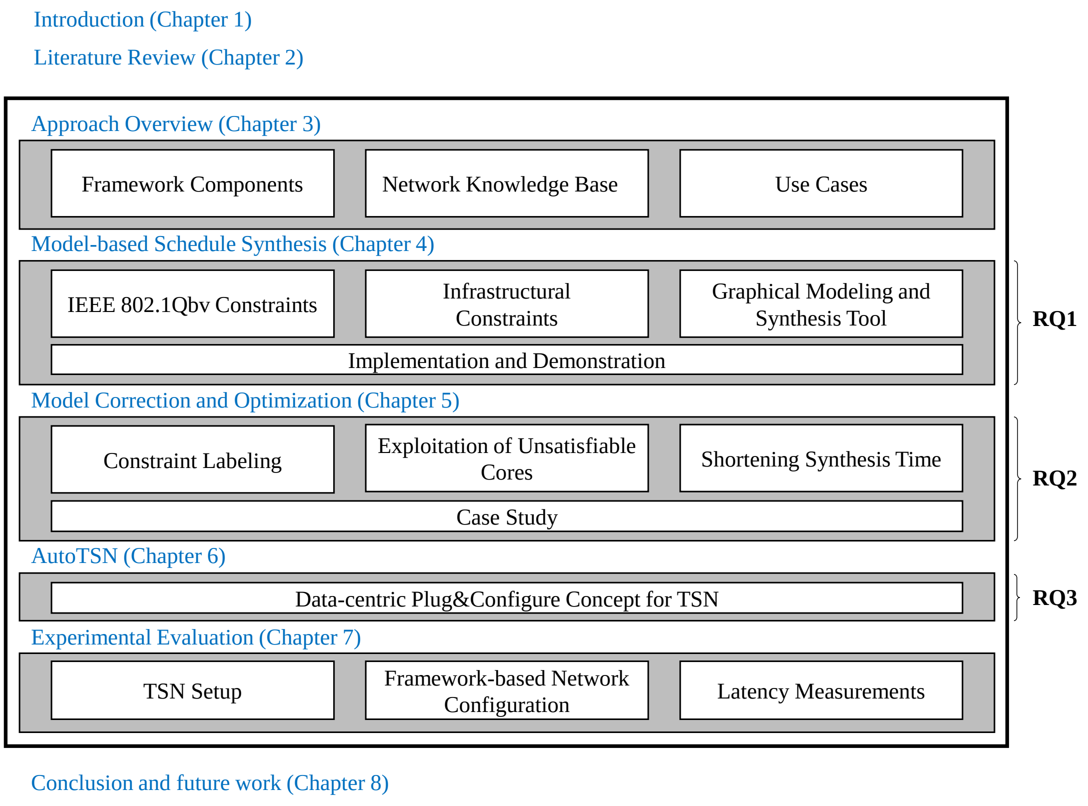

Introduction (Chapter 1)
Literature Review (Chapter 2)
Approach Overview (Chapter 3)
Framework Components
Network Knowledge Base
Use Cases
Model-based Schedule Synthesis (Chapter 4)
IEEE 802.1Qbv Constraints
Infrastructural Constraints
Graphical Modeling and Synthesis Tool
Implementation and Demonstration
RQ1
Model Correction and Optimization (Chapter 5)
Constraint Labeling
Exploitation of Unsatisfiable Cores
Shortening Synthesis Time
Case Study
RQ2
AutoTSN (Chapter 6)
Data-centric Plug&Configure Concept for TSN
RQ3
Experimental Evaluation (Chapter 7)
TSN Setup
Framework-based Network Configuration
Latency Measurements
Conclusion and future work (Chapter 8)

Figure 1.4: Thesis Structure

1.4 Structure

The thesis is structured as follows. In Chapter 2, the state of the art is reviewed. The focus is on most recent achievements in time-triggered scheduling theory, formal timing analysis of Ethernet-based networks, Plug&Play approaches for industrial Ethernet, and model-based network engineering concepts. Research gaps are identified and presented which motivate the contributions of this thesis.

The modeling framework and its main components are presented in Chapter 3. The model-based schedule synthesis approach is explained in Chapter 4. Model correction and optimization are presented in Chapter 5 explaining the overall concept based on a constraint labeling approach and exploitation of the state of the art in the extraction of minimal unsatisfiable cores. The developed Plug&Configure concept AutoTSN is discussed in Chapter 6. The experimental setup and benchmarks are placed in Chapter 7 which is used to demonstrate parts of the developed concepts using prototypical switches supporting synchronization and IEEE 802.1Qbv.

Finally, we conclude in Chapter 8 and discuss the planned future works. An overview of the thesis structure is depicted in Figure 1.4.

Chapter 2

Literature Review

In this chapter, the related work is studied and analyzed based on the research questions. We aim to derive concrete required contributions which help to answer the research questions. Time-triggered schedule synthesis, timing analysis of TSN, autoconfiguration, and network modeling approaches are reviewed and summarized at the end of the chapter in Table 2.1

2.1 Schedule Synthesis for Time-triggered Ethernet

Basic time-triggered constraints such as the non-overlapping of Ethernet frames are formulated. The experiments using the YICES solver which is based on Satisfiability Modulo Theories (SMT) show that the scheduling problem can be solved for a few hundred random stream instances. Regarding the scheduling quality, the approach produces up to ninety percent maximum utilization on a network link with synthesis times of about thirty minutes for the biggest study case.

In a further work [25], the co-existence of time-triggered traffic with non-time-triggered traffic is discussed. The authors introduce the term *schedule porosity* and study the impact of time-triggered traffic on unsynchronized traffic. The core idea is to enlarge the blank space between time-triggered slots to transmit the non-synchronized traffic.

The concept of creating communication schedules for the time-triggered traffic while minimizing its makespan is studied in [26]. In contrast to [25], the proposed approach, maximizes the uninterrupted gap for non-synchronized traffic in each integration cycle. The approach is based on a load balancing heuristic. The results show enhancements in comparison to the schedule porosity approach by analyzing the worst-case delay of non-synchronized traffic.

In [27], the authors study time-triggered distributed systems where periodic application tasks are mapped onto different end stations which communicate over an Ethernet network. As distinguished to [28], application-level constraints are studied. The co-synthesis problem of task and communication schedules is formulated as a Mixed Integer Programming (MIP) problem considering different timing parameters such as precision of synchronization. The authors present optimization objectives regarding timing. The applicability of the approach is shown considering an industrial size case study using a number of different sets of objectives.

The simultaneous co-generation of static network and task schedules is discussed in [29] with a focus on preemptive time-triggered tasks (in contrast to [27]) communicating over switched time-triggered networks. The authors use first-order logical constraints for the formulation of schedule constraints which are solved by SMT and MIP solvers. Furthermore, asynchronous tasks have been considered to improve the scalability of the scheduling problem. Based on the evaluation results, the authors conclude that even if using optimization, most of the problems are solved within a reasonable time using the developed method.

This work is further extended by [30], who identifies and analyzes key functional parameters affecting the deterministic behavior of real-time communication under IEEE 802.1Qbv. The required constraints are derived (based on [28]) to compute schedules guaranteeing deterministic low-jitter and end-to-end latency for time-triggered streams. Moreover, several optimization directions and concrete configurations exposing trade-offs against the required computation time are discussed. The significant contribution is the introduction of two additional isolation constraints which are required to consider the order of arriving critical frames at the ingress ports.

The time-triggered schedule synthesis is also discussed in [31]. Their main objective is to make the SMT solving efficiently regarding run-time and memory. Moreover, an approach is proposed to parallelize the synthesis problem. The authors conclude that the solver YICES2 is an effective alternative tool compared to the available SMT tools.

An alternative synthesis approach using memetic-based algorithms is presented in [32]. It derives a feasible schedule by determining the offset of frames on the time-triggered network-on-chip. The approach is based on a memetic algorithm that incorporates local search in the iterations of a general genetic algorithm. The results are not compared with the existing related work using SMT solvers.

An offline synthesis tool is presented in [33], which tackles the challenge of long synthesis time. It is capable of computing time-triggered schedules for large-scale networks consisting of wired and wireless links. In this approach, linear arithmetic constraints are used to model the problem and are solved using a SMT solver. The total problem is divided into solvable scheduling problems with smaller size. The sub-solutions are combined to achieve the overall schedule. The authors discuss optimizations to increase the size of the solvable scheduling problems. The evaluation of the proposed approach shows that the synthesis time is reduced.

The routing optimization and worst-case delay analysis of AVB streams in TSN networks are discussed in [34]. They aim at finding the optimal routing of the AVB streams so that the set of all frames is feasible. Moreover, the worst-case end-to-end delay is defined as an optimization objective which has to be minimized. The proposed approach contains a search-space reduction technique based on a Greedy Randomized Adaptive Search Procedure (GRASP) heuristic to formulate and solve the optimization problem.

The authors of [35] criticize that in existing network design methods, stream routing and schedules are generated in two separated steps without considering the interrelations between them. An exact approach is proposed to generate routing and schedules in one step. To achieve this, the problem is formulated as a 0-1 ILP. Moreover, [35] shows that the formulation can be used to optimize the routing and scheduling considering the impact on non-scheduled traffic or the number of slots which have to be configured on a port. In contrast to this work, we apply a model verification prior to synthesis to ensure that the number of port slots matches to the hardware properties of the port.

In a similar work [36], a survey regarding the optimization of networks based on real-time Ethernet is presented. Moreover, optimization problems related to the scheduling and routing of time-triggered and AVB traffic in TSN are formulated. The main objective is to compute the routing of both time-triggered and AVB streams as well as the scheduling of the time-triggered streams. Constraints are built to guarantee, that all frames are schedulable and the AVB worst-case end-to-end delay is minimized. The scheduling problem is formulated in ILP and for the routing problem, a greedy randomized search heuristic is applied. Routing and scheduling of TSN streams are also discussed in [37, 38] formulated as SMT and ILP problems.

Formulation of scheduling problem as a system of constraints is not only applied for Ethernet networks. An approach based on boolean satisfiability (SAT) problem is presented to tackle the problem of task and message allocation of distributed real-time systems [39]. The problem is formulated as a nonlinear integer optimization problem. The authors claim that the presented method is applicable to industrial-size task systems.

A scheduling approach for the synthesis of time-triggered embedded systems is presented by [40]. A priority function is introduced which considers the communication protocol. The authors present another approach in [41] to support static priority preemptive process scheduling for hard real-time time-triggered communication. The communication delays with different message scheduling policies over a time-triggered communication channel are studied.

The schedule synthesis for PROFINET IRT is discussed in [42]. The authors formulate the scheduling problem as ILP. The results show that the approach is slower than SIMATIC (a commercial tool developed by SIEMENS [43] for optimizing the engineering workflow) but is more flexible in terms of specifying the application requirements.

In [44], the schedule synthesis of the time-triggered communication protocol FlexRay [45] is discussed. FlexRay is not based on the Ethernet and supports the transmission of periodic messages in static segments. Priority-based scheduling of event-triggered messages is handled in dynamic segments. The authors aim at optimizing the performance of application-related timing metrics. The scheduling problem is formulated as Mixed Integer Linear Programming (MILP).

A framework for automotive architectures is introduced in [46] using data-centricity and time-triggered communication paradigms. Similar to the incremental method of [28], local schedules are first defined independently for subsystems which are further integrated into a system-wide schedule. The main objective is to reduce integration complexity. A contribution by [47] based on the proposed approach, formulates the scheduling problem of FlexRay in ILP. The focus of this work is the automotive domain and schedule synthesis for different vehicle variants.

Summary: The reviewed approaches present the state of the art in schedule synthesis for time-triggered Ethernet. We use the existing scheduling constraints and extend them by infrastructural constraints to verify that a network model satisfies all prerequisites for the synthesis. For instance, we must verify that the maximum allowed number of GCL entries of a switch port is not exceeded and also the synchronization capability which is required for IEEE 802.1Qbv is available for ports which are involved in the transmission of time-triggered streams. Such requirements have to be verified before synthesis is started.

Moreover, and in contrast to this thesis, the mentioned contributions do not deal with comprehensible and user-friendly network modeling (graphical elements to hide details). The existing approaches do not address the question of how to extract relevant information from the model to generate solver-specific constraints automatically. Additionally, we will generate feedback for the designer to correct the network model if it is unsatisfiable because of conflicting constraints.

2.2 Timing Analysis of AVB/TSN Networks

Besides the synthesis approaches which aim at finding a feasible schedule and a concrete network configuration, there are contributions which deal with timing analysis of AVB and TSN networks. The main objective is to find an upper bound for end-to-end latency and jitter of the streams. These contributions can be divided into two categories. The first category contains contributions that apply network simulation for timing analysis. The second category incorporates contributions where a formal analysis method is used to derive formally verified worst-case latency and jitter.

Considering the simulation-based approaches, a major contribution is the extension of OMNeT++ INET [48] to simulate real-time Ethernet [49]. The time-triggered Ethernet protocol is implemented and used to evaluate the timing of in-vehicle networks.

Performance comparison of AVB and IEEE 802.1Q is presented by [50], which focuses on the automotive network architectures. The simulation results show that AVB improves the real-time capabilities of Ethernet but it still requires advanced features to fulfill the tight timing requirements from the automotive domain especially related to the critical control data where low deterministic end-to-end latency and jitter are required.

AVB and time-triggered Ethernet are analyzed and compared regarding their performance with a focus on driver assistance systems [51]. The authors conclude that the two protocols are complementary. The time-triggered streams have to be scheduled offline and the AVB allows dynamic stream reservation at run-time. A similar competitive evaluation of AVB and time-triggered Ethernet is presented in [52] with similar results.

The coexistence of scheduled and non-scheduled traffic is studied in [53], where the time-triggered frames are added to the credit-based shaper of AVB. The simulative results show that the time-triggered schedule has a significant impact on AVB messages. The available bandwidth between reserved time-triggered slots has to be large enough for sending of AVB frames in order to fulfill their timing requirements. The simulations show that with smaller MTU size, delays of AVB streams can be reduced.

A comparable study [54] shows that event-based traffic classes can increase the end-to-end latency of AVB streams by 500% in the network. To reduce this negative impacts, the authors propose the following measures: limiting the MTU size, limiting the event-based traffic, increasing the bandwidth of the network, optimizing topology, and supporting of frame preemption mechanisms. The optimal MTU size is studied in [55] by simulating different cases to find the optimal bandwidth which has to be allocated to the scheduled traffic.

A simulation framework is presented in [56] to evaluate a case study considering the frame preemption and per-stream policing standards. Based on the achieved results, the advantages of frame preemption highly depend on the network design and configuration.

Simulations are helpful to understand the abstract properties of a system but they do not give any guarantees that all corner cases are explored. For safety and critical systems, formal analysis methods are required which can deliver mathematically proved guarantees.

Considering the formal analysis approaches, a formal worst-case analysis is presented by [57]. The study examines the timing of Ethernet AVB and strict-priority Ethernet IEEE 802.1Q. Safe upper bounds of stream transmission are determined using Compositional Performance Analysis (CPA) [58]. CPA is the theoretical background of the commercial tool SymTA/S. Based on case studies including star, line, and clustered line topologies, the approach is evaluated. The results show that by applying Ethernet AVB, the latencies for Class A traffic is substantially increased compared to the strict-priority mechanism in IEEE 802.1Q. The authors explain this negative effect by mentioning the traffic shaping delay.

Using network calculus, the worst-case delay of AVB streams is analyzed in [59]. Based on an automotive case study, the authors conclude that network calculus is a suitable estimation method for buffer and timing analysis in the automotive domain. However, the service and arrival curves have to be optimized to reduce the pessimism of the analysis.

Another timing analysis of AVB is presented in [60], which is based on the Real-Time Calculus (RTC) [61] methodology. The results show that a well-designed AVB network performs better than standard Ethernet. In contrast to [57], the negative impact of shaping delays can not be observed in the results.

The schedulability analysis of AVB is discussed in [62], where the authors aim at improving the results achieved by [57]. The contribution of this paper is to consider AVB specific challenges such as the number of blocks which is caused by low-priority frames and the state of the shaper. Another improvement of AVB performance analysis is introduced in [63], where non-preemptivness is considered for computing tight latency bounds. The experimental results show an enhancement of 50% for a line topology.

The authors of [64] mention the challenge of obtaining tight upper bounds for AVB streams ifthe busy period analysis is applied. The reason is that the busy period method is based on execution time without considering the provided bandwidth. An alternative method is proposed by defining a so-called eligible interval.

The timing analysis of time-triggered Ethernet frames and rate-constrained traffic [2] is significant in the context of performance analysis of TSN and AVB because these paradigms are similar. IEEE 802.1Qbv defines a time-triggered approach similar to time-triggered Ethernet. The credit-based shaper of the AVB standard specified in IEEE 802.1Qav has also similarities to a rate-constrained approach.

Performance analysis of time-triggered Ethernet in the combination of rate-constrained traffic is presented based on RTC by [65]. The experimental results show that the proposed approach is able to achieve tighter upper bounds for worst-case delays. In [66], a novel worst-case end-to-end delay analysis of the rate-constrained traffic is presented. The main objective is to reduce the pessimism of the analysis, compared to existing approaches. The major contribution is to compute the busy period using the concept of event-triggered availability and demand [67]. The results are compared to the achievements of [25] and show considerable improvements.

Worst-case analysis of TSN traffic shapers (time-aware, peristaltic and burst-limiting shapers) has been recently presented [68]. Based on the results, only the time-aware shaper can support very low latency guarantees. The authors claim that peristaltic and burst-limiting shapers can be enhanced using the frame preemption mechanism (under development in IEEE 802.1Qbu). This approach has been improved in [69] using CPA by considering the interference of same-priority frames. The authors assume that network nodes are not synchronized and calculate the worst-case latency and jitter of the time-aware shaper. The applied formulation regarding the busy period calculation is similar to [66]. The results show that the time-aware shaper achieves the lowest latency only if network nodes are synchronized.

The impact of frame preemption in TSN networks is formulated and studied in [70] using the CPA tool. The proposed approach is used to evaluate an exemplary automotive network. The results show that with frame preemption the performance of strict-priority traffic shaping specified in IEEE 802.1Q is dramatically improved and is very close to the TSN time-aware shaper. The authors conclude that for common automotive network architectures, the latency and jitter performance of the standard Ethernet may be sufficient. However, this conclusion can not be generalized because the required timing guarantees always depend on concrete application requirements.

The suitability of SDN for TSN networks is discussed in [71]. Based on CPA method, the authors analyze the impact of SDN configuration messages on real-time traffic and vice versa. The proposed approach is evaluated using a typical automotive network. The results show that the worst-case network configuration latency is less than $50ms$ and that SDN can be used for routing control in fail-operational use cases.

The impact of TSN time-triggered schedules on non-scheduled traffic is discussed in [72]. The presented method does not make any assumptions regarding the shape of the time-triggered slots. This leads to long run-times. The authors present a method based on preprocessing and extraction of related data for the worst-case timing analysis. Based on a case study, the authors conclude that the time-triggered schedule on egress ports plays a significant role regarding the delay of the unscheduled traffic and has to be considered at the network design time.

Similarly, [73] presents a local delay analysis of AVB frames for which both credit-based and time-aware shaper are used. The authors study how the time-aware shaper changes the relative order of transmission of AVB frames which causes bursts and worst-case for frames with lower priority. However, it is also shown that the resulting bursts are upper-bounded by the credit-based shaper. Schedulability analysis for the real-time traffic in AVB ST [74] networks is presented in [75]. The AVB ST is an improvement of AVB standards which is a similar concept to IEEE 802.1Qbv supporting scheduled traffic. The authors show that bandwidth reservation based on the AVB standard increases the pessimism considerably and therefore, a bandwidth over-reservation is needed. Moreover, a minimized bandwidth over-reservation approach is proposed. The proposed approach is evaluated using an automotive case study introduced by [76].

A recently published work [77] presents empirical evaluations of IEEE 802.1Q, AVB, and TSN based on an automotive prototype. Different network combinations of communication technologies are presented and compared. The main comparison parameters are the end-to-end latency and verification overhead of each possible combination. The authors conclude that the following solutions are suitable for automotive: standard IEEE 802.1Q with a pre-shaping of the traffic, AVB with custom classes and tight idle-slope, and, finally, synchronized TSN with AVB custom classes. The configuration of TSN gates is mentioned as a challenge that requires more support for automated procedures. This statement additionally supports the motivation of this thesis.

Summary: The simulation-based timing analysis approaches can be used for the analysis of those streams without safety-relevant timing requirements such as infotainment streams.

Because simulations do not guarantee corner worst-cases, they can not be used for safety-critical streams. Formal timing analysis methods can be used to compute verified latency upper-bounds for critical streams. These approaches are known to be pessimistic [78] and computationally intensive [66]. One of the reasons for the pessimistic results is the fact that the concrete network configurations such as TSN schedules on egress ports are not considered in the analysis. In the reviewed formal approaches, proposed network models (except the UML diagrams in [79]) focus more on the analysis method itself (e. g. arrival or resource curves [61]). In contrast, we aim to hide such details in our framework applying a user-friendly graphical modeling tool. Network verification regarding the infrastructural requirements of IEEE 802.1Qbv is not addressed in the literature.

2.3 Autoconfiguration Concepts for Real-time Ethernet

The term Plug&Produce describes a methodology which aims at introducing a new manufacturing device into an existing manufacturing system easily and quickly [80, 81]. It is based on a similar idea of plug&play in the IT world. One of the significant requirements to support Plug&Produce is the automation of the network configuration to increase the system flexibility at run-time. To achieve this, the underlying communication technology (e. g. industrial Ethernet) and also the timing requirements from the application layer have to be considered in the automated configuration procedure. In the following, we review contributions which aim to reduce the configuration overhead of the Ethernet-based real-time networks.

Opportunities and challenges regarding the dynamic configuration of industrial automation networks are discussed in [82]. The authors illustrate the capabilities of service-oriented architectures such as Device Profile for Web Services (DPWS) to increase the flexibility in such networks. Different sample architectures based on Programmable Logic Controllers (PLC) are used to demonstrate the concepts and ideas.

The missing flexibility in real-time Ethernet networks was the motivation to develop the FFT-Ethernet protocol which is presented in [83, 84]. It provides a master and multi slave transmission control method and guarantees that timing requirements are fulfilled while dynamic configuration activities are executed. The slave architecture contains a real-time and a non-real-time stack which supports user tasks without timing requirements. The FTT-Ethernet frames are embedded in a standard Ethernet frame.

A five-step-model for the reconfiguration of Ethernet Powerlink is presented in [85]. The device profiles of controlled and managing nodes are used in a control station to compute the required configuration parameters. The proposed approach depends on specific properties of the Ethernet Powerlink protocol and cannot be generalized for other industrial Ethernet technologies.

The approach presented in [86] proposes an autoconfiguration solution for Profinet IO which is based on DPWS. The presented approach acts as an adapter for native Profinet IO devices and controllers. The configuration and mapping services are located in the controller devices. The configuration service is responsible to parse the received device descriptions. The mapping service writes the configuration parameters in the Profinet IO stack which is a similar procedure to [85]. The authors conclude that the approach has to be extended to consider higher-level control applications. A further contribution in [87] considers the OPC Unified Architecture (UA) [88] as a suitable candidate for the autoconfiguration of real-time Ethernet networks. The authors mention that using OPC UA is a good choice for devices with limited computing resources. Similar to [86], a Profinet IO setup is used to evaluate the presented concept. A concept is introduced to increase the adaptability of IT systems for automation use cases based on a model-based Plug&Play approach [89]. The proposed concept aims at reducing changeover time and configuration efforts. To achieve this a data-centric middleware called CHROMOSOME [90] is applied.

The configuration complexity and the requirements of future factories are discussed in [91]. The authors developed a system architecture to automate the integration of new devices and network configuration based on OPC UA. Three general requirements are derived to support Plug&Produce: (i) Communication links have to be set up automatically and the controller has to be informed, (ii) the communication data (which is defined at the application layer) has to automatically be configured for devices, and (iii) the configuration has to be distributed among the network nodes using a software component. The proposed approach is evaluated using a prototypical setup based on EtherCAT.

With regards to the challenge of dynamic reconfiguration of automation networks, a single-master and a multi-master approach are presented in [92]. The approach is based on an extension of the EtherCAT state machine and aims at increasing the flexibility of the configuration procedures. The authors mention that the single-master approach offers the advantage that no user manipulations are required. However, each device requires a modification of the device profile which is challenging and currently not defined in the device descriptions. The multi-master solution leads to a shorter reconfiguration time in comparison to the single-master approach.

The mentioned approaches do not fully address the generalizability issue which is also mentioned in [93]. The authors categorize the existing approaches into three groups and derive general functionalities which are required for autoconfiguration of real-time Ethernet networks. The result is a four-phase procedure including IP-Connectivity to prepare the configuration procedure, Device Discovery to register new devices in the network, Requesting real-time-specific parameters, and uploading the new configuration to the controller. The authors also mention the need for investigation of TSN standards which will have significant impacts on the proposed approach.

A protocol and architecture are proposed to enable real-time Ethernet networks for self-configuration based on OpenFlow [94]. The focus here is on communication link failure recovery which is similar to the objectives of IEEE 802.1CB standard. The authors conclude that the proposed approach can be adapted to TSN networks. Using randomly generated topologies, the proposed architecture is evaluated.

Software-Defined Networking (SDN) [95] is a paradigm towards automated network configuration. In SDN terminology, there are two planes. The data plane deals with forwarding of Ethernet frames and control plane is a logically centralized entity which controls the data plane based on the network application flows. Considering the TSN standard, SDN concepts have significant relations to IEEE 802.1Qcc where stream reservation and User Network Interface (UNI) are discussed.

A three-step agent-based configuration approach for time-triggered networks is proposed by [96]. In the first step, the traffic is observed to identify traffic patterns. These patterns are characterized using the id, length, and stream arrival times. In the next step, the traffic parameters are obtained using an extractor entity. The traffic parameters contain information about the periods of the frames, for example, and their priorities (cf. VLAN tag in IEEE 802.1Q) and a possible precedence order of the frames. Finally, the extracted information is used to compute schedules. The authors plan to evaluate the approach based on simulations. The proposed approach is further extended in [97] where network management of TSN networks is discussed in details with a focus on the running activities in IEEE 802.1Qcc and SDN concepts.

Considering the configuration of real-time Ethernet networks, each communication technology requires its proprietary configuration and programming tools. Examples are TwinCAT [98] and Automation Studio [99], which are used to configure and program EtherCAT and Ethernet

Powerlink networks. These tools are used to simplify the configuration procedures using graphical user interfaces where configuration parameters such as communication are set by control engineers.

currently, the first configuration tools for TSN networks are under development. HiVision [100] is used to detect connected nodes and build a graphical network topology based on the obtained information. It offers an interface to set the GCL entries of egress ports. However, time-triggered schedule synthesis is not supported. It is a pure configuration tool which requires manual entries calculated by the network designers or control engineers.

Slate XNS [101] is the nearest configuration tool to the developed modeling and synthesis tool in this thesis. It is under development and aims to compute the time-triggered schedules for TSN ports using a graphical modeling tool. Based on the latest product description, the modeling of network topology and streams will be supported and the computed schedules will be distributed among switches and end-stations. In contrast, our modeling tool computes the streams automatically based on a high-level and data-centric description of the applications. Moreover, the proposed Prolog-based queries in this thesis enables the verification of the infrastructural requirements. For example, the existence of the redundant paths to support fail-operational or not exceeding the maximum number of allowed GCL entries can be verified before starting the synthesis. Additionally, in the case of model unsatisfiability, our modeling tool can produce meaningful feedback pointing to concrete hardware and applications helping the designers to resolve conflicting scheduling constraints.

Summary: The discussed approaches focus on the Ethernet-based field bus systems with PLCs acting as masters. PLCs are therefore responsible for application management and controlling the communication in the network. Because these systems are proprietary, the proposed autoconfiguration approaches are also significantly system-dependent when it comes to implementation. In contrast, network configuration task in TSN is done within network components such as switches and Network Interface Controllers (NIC) of the end-stations. Although the application requirements such as period and data size are relevant for correct network synthesis and configuration, the configuration procedure itself is technologically separated from the application layer. This kind of decoupling promotes modularity and increases the capability for a generic autoconfiguration approach by exploiting modern middleware paradigms and technologies.

2.4 Modeling of Real-time Networks

Network modeling is used to facilitate the specification, verification, simulation, and configuration of computer networks. Different simulation examples have been presented in Section 2.2. These simulation tools use graphical modeling elements of e. g. OMNET++ INET framework to model nodes, links, and protocols. There are modeling approaches which deal with real-time networks focusing on specification and verification.

To characterize the automotive software architecture and to distribute processes and tasks on processors, a graphical notation is introduced by [102]. The presented tool (AutoFocus) is based on a client-server paradigm using a central repository and distributed client applications. The dynamic behavior of the network components is described using state transition diagrams. This tool is based on the Focus framework [103] which is developed for formal specification and verification of interactive systems.

UPPAAL [104] is a toolbox based on timed automata to support modeling, simulation, and verification of real-time systems. It offers graphical and textual modeling elements. It contains three main parts for (i) a description language, which is used to describe network behavior based on timed automata, (ii) a simulator, and (iii) a model checker which supports an interactive analysis of the system behavior. To achieve this, the constraints are solved which describe the state space of a system description. UPPAAL is one of the standard tools widely used for exact timing analysis of distributed applications for example in the domains of industrial automation [105, 106] and avionics [107].

COMET is a methodology to design real-time and distributed applications [108]. It integrates object-oriented and concurrency concepts using Unified Modeling Language (UML) [109]. Following the COMET methodology, static and dynamic models of a system are developed in the analysis modeling phase. Furthermore, an architectural design model is developed in the design modeling phase.

Ptolemy [110] is a simulation and prototyping environment for heterogeneous systems. It has an object-oriented and modular design allowing the integration of subsystems. It supports a graphical user interface to model concurrent components.

Stateflow [111] is a graphical modeling and simulation tool which can be used to model reactive systems. The modeling elements are based on state machines and flow charts available in MATLAB Simulink. SysML [112] is a graphical modeling language based on a subset of UML.

It is used for system engineering and supports mechanisms for system design and verification. It can be used for the modeling of embedded systems regarding timing [113].

Based on RTC [61], [114] proposes a tool to model and analyze distributed networks with real-time requirements. It is implemented as a MATLAB Simulink toolbox and implements min-plus and max-plus algebra operators required for modeling of task and resource curves. Similarly, a graphical modeling and performance analysis tool is proposed in [115] based on deterministic network calculus and is implemented for the MATLAB Simulink environment. It aims at providing solutions for systems containing cyclic dependencies. A set of scheduling and arbitration mechanisms is supported such as: FIFO, TDMA, round robin, EDF, and fixed priorities.

A modeling approach is presented for Ethernet AVB networks in [79]. Using UML class diagrams, network elements such as switches, stations, and Ethernet ports are modeled. The main intention of [79] is to use the presented metamodel to design model instances for formal timing analysis based on CPA. Hence, a model transformation approach is introduced which prepare the UML-based AVB network models for the CPA platform. In contrast to this contribution, the metamodel presented in this thesis follows the data-centric publisher-subscriber paradigm and offers a graphical editor providing drag&drop features to achieve a high degree of user-friendliness.

Graphical visualization concepts using comprehensive notations increase the analyzability of complex real-time networks (e. g. the industrial automation domain). A graphical modeling approach is proposed in [116] to model Multicore Programmable Logic Controllers (MPLC) in networked automation systems to increase the analyzability of these systems. This contribution is an extension of the notation presented in [117] focusing on properties and requirements of switched automation networks.

Modeling approaches are also used to design and configure dynamic networks. Based on a data-centric paradigm, [90] proposes the middleware CHROMOSOME. It consists of a run-time environment XME and a modeling tool XMT. The run-time environment consists of (i) a *Data Handler* that offers an API to the applications to publish and subscribe their data in the network, (ii) a *Broker* that monitors the availability of the data in the Data Handler and checks if the QoS requirements are fulfilled, and (iii) an *Execution Manager* that executes the applications which are enabled by the broker. To simplify the application development and automated code generation, graphical XMT can be used. Moreover, it provides the capability of specifying timing requirements for correct initializing of the XME. Except for the modeling tool, the concepts used in this approach are similar to other data-centric middleware implementations [118, 119].

A tool for compositional analysis of real-time systems is presented in [120]. The authors mention the importance of complexity management for high-assurance and cost-effective development. The main objective is to divide a system into smaller components which can be independently developed reducing the complexity [121]. A tool is implemented in [121] based on Ptolemy [110] libraries and offers a graphical user interface to model e. g. resources and tasks demand. The authors address the problem of compositional real-time guarantees in the hierarchy of schedulers.

Recently, approaches have been presented which exploit the capabilities of logic programming paradigm for modeling, specification and verification of Cyber-Physical Systems (CPS). Based on the Constraint Logic Programming (CLP) concept, a framework is presented in [122] to specify and verify CPS with timing requirements. The main objective is to bridge between logic programming and hybrid automata as the underlying model. Different properties of a real-time system can be verified using adequate queries against the CLP model. Moreover, novel types of formalisms called constraint automata and timed pushdown automata are proposed by the authors. The main ideas of the presented approach are further developed by [123, 124] who evaluate the modeling approach using a temperature control system of a reactor as a case study. The authors demonstrate how queries can be used to verify the specific properties of the system.

Designing and adapting of overlay networks to desired application domains such as storage systems and communication infrastructures, is a complex and time-consuming task for network managers [125]. Based on this motivation, a methodology is developed to formulate overlay networks using a variant of the Datalog (declarative logic language) called OverLog. The applicability of the approach is demonstrated using two case studies: (i) Narada multicast functionality, and (ii) Chord distributed lookup service with 15 OverLog rules. The authors additionally evaluate the timing performance of the presented approach using the Emulab network testbed.

The concept of declarative routing is presented in [126]. The authors aim to balance between the extensibility and robustness of routing infrastructures. The main objective is to express routing protocols based on a database query language. The computational expressive power, the security aspects, and the language design of the proposed approach are discussed. Using simulations and based on PlanetLab [127], it is shown that the approach is applicable to express routing protocols and can be extended for query optimizations.

A declarative network modeling approach is proposed in [128] based on Datalog. The Network Datalog (NDlog) language for declarative network specifications is introduced and formally defined. Using a path-vector protocol example, the modeling approach is demonstrated. Moreover, NDlog is extended with link-restricted rules to capture physical network constraints. NDlog is also used for declarative network verification of e. g. routing protocols as proposed in [129]. The authors present the design and development of the Declarative Network Verifier (DNV). It takes a declarative model of a network protocol written in DNlog as input and converts it to logical axioms which can be used for formal verification. The main objective is to reduce the effort required for network and system specification.

A similar concept is proposed in [130] where a declarative policy language for managing the configuration of computer networks is presented. The Flow-based Management Language (FML) is a restricted form of Datalog and aims to replace the traditional configuration mechanisms to enforce network policies. Example use cases are VLANs and policy-routing where FML can be used to express the policies. Using empirical studies, the authors demonstrate the performance of the developed approach.

In [131], it is shown that Prolog is sufficiently expressive and suitable for the implementation of distributed protocols. The authors claim that the declarative programming paradigm has a promising potential to support developers of distributed systems. A Prolog-based programming system called DAHL is presented which extends Prolog with an event-driven control mechanism and built-in networking procedures. The presented system is evaluated by implementing (i) a distributed hash-table data structure, (ii) a protocol to provide Byzantine fault tolerance, and (iii) a model-checker in DAHL as case studies to ensure the approach feasibility.

The complex relations and possible conflicts between application requirements in automotive architectures are taken as motivation to develop a formal and Prolog-based approach for network specification and verification [132]. The proposed approach is useful when e. g. an engineer has to carefully analyze the possible effects of adding new sensors or actuators on other existing critical applications with timing, bandwidth, fail-operational, or routing requirements and constraints. The network has to be configured such that the fulfilling of all requirements is guaranteed. The main focus is on TSN where logical facts and rules are formally defined to develop adequate verification queries. The exploitability of the approach is demonstrated using exemplary queries to guarantee the existence of redundant and disjoint network paths. Based on the results, a graphical modeling tool for automated schedule synthesis in TSN networks is developed in [133]. These two contributions build together the basis of the modeling approach of this thesis.

Ontology-based approaches are also applied for network management tasks [134–137]. These are relevant because ontologies and logic programming have significant similarities considering mathematical logic which builds the basis for ontology reasoners and solving of the Horn clauses (cf. Prolog). Advantages and shortcomings of the ontology-based approaches are investigated by [138]. The paper claims that ontology-based interoperability frameworks can help to automate several tasks in network management.

Descriptive network management policies (e. g. firewall rules) are proposed in [139] and [140] to show the capabilities of ontology-based network management. An ontology-based information extraction system is presented in [141] to fill the configuration gap in hybrid SDN. One use case is the formalization of switch and router configuration to reduce the configuration overhead for administrators.

An ontology-based approach is proposed in [142] to improve the modeling and plug-and-play capabilities of TSN. The main idea is to use the capabilities of ontology tools for modeling data-centric networks. The approach is demonstrated using an automotive case study. Because of limited and abstract graphical features of existing ontology tools which did not allow to model the network topologies and dataflows for schedule synthesis in a user-friendly way, a new solution is presented in [132, 133] to combine Prolog (as an inference engine) with a powerful graphical modeling tool.

Summary: Simulation frameworks are used to model a network and to analyze specific properties such as timing behavior. Graphical modeling elements are available e. g. in OMNET++ INET to create user-friendly network models. However, such graphical models cannot be used for infrastructural model verification or for the extraction of relevant information to generate schedule constraints. The same argumentation is valid for tools which are used for formal timing analysis. Either there is no adequate graphical modeler (cf. RTC [114]) or the provided graphical modeler is developed for a very specific domain (cf. UPPAAL [104]) and cannot be used for modeling and automation of schedule synthesis. Moreover, the data-centricity paradigm is not considered in the proposed approaches (except [90] and our previous contribution presented in [133]).

The declarative networking approaches which exploit the logic programming paradigm support network model verification especially regarding the infrastructural requirements. Regarding schedule synthesis, generated network facts and pre-defined inference rules (we call this network *knowledge base*) can be used as an effective instrument to query relevant information for enabling automated schedule synthesis. Moreover, such a network knowledge base plays an important role in making configuration decisions at run-time (e. g. whether a new synthesis is required when a new device joins the network). There is a lack of user-friendly modeling approaches which can be used to automate the process of generating facts to build the desired network knowledge base. The presented approaches also do not provide any solution for generating correction feedback of unsatisfiable models to help the network designers.

2.5 Discussion

Motivated by the introduced research questions (automated configuration, verification, and at run-time re-configuration), a set of features to facilitate TSN-based developments are identified. In the following, we explain these features more precisely and use them to evaluate the related work.

To simplify the design process of TSN-based network architectures, a **Graphical Data-centric Network Modeler** feature is beneficial. It has to provide modeling elements for the topology, logical dataflow, and timing requirements of applications. The support of data-centricity (publisher-subscriber paradigm) increases the flexibility of modeling such that static definition of streams (including the sending and receiving end-stations and affected egress ports on the path) are no longer required and can be concluded automatically by the modeler. This is achieved by analyzing the publisher-subscriber relations and the network topology.

A **Network Knowledge Base** can considerably facilitate the information management required for configuration and verification of TSN networks. For instance, it can be used to infer the topological relevant egress ports and timing requirements of a certain real-time application to derive schedule constraints. Moreover, it can be used to verify network properties such as available redundant paths and synchronization capabilities.

Building the time-triggered constraints for schedule synthesis is a complex process. To fully automate this process, a feature (**Model-based Generated TT Constraints**) is required that handles the high-level graphical network models (made by network designers) and uses them to (i) build a network knowledge base, (ii) generate the time-triggered schedule constraints, (iii) solve the constraints to find feasible schedules (**TT Schedule Synthesis**), and (iv) distribute the schedules to switches and end-stations.

There are different approaches to solve a constraint-based schedule synthesis problem. An assisting solution has to provide an **Infeasibility Analysis** feature which can be used to back-trace the conflicting constraints. Moreover, a (**Correction Assistance**) feature is required to label the constraints with high-level network information (including e. g. topology and applications to localize the origin of schedule infeasibility). Moreover, it has to find minimal unsatisfiable constraint subsets such that efficient model correction activities (e. g. relieving an overloaded egress port) can be concluded.

To find worst-case latency and jitter values of the modeled streams, **Formal Timing Analysis** and **Simulation** features are required. Before starting the scheduling synthesis based on IEEE 802.1Qbv, it must be verified that the prerequisites for such synthesis are fulfilled (e. g. switch ports support synchronization and do not exceed the maximum number of GCL entries). An **Infrastructural Verification** feature for IEEE 802.1Qbv is required which exploits the network knowledge base and supports pre-synthesis verification queries.

To enable TSN for highly dynamic networking with hard real-time guarantees, a feature (can also be used at design time) is required at run-time that analyzes the application-layer modifications (e. g. a new end-station joins the network which publishes hard real-time periodic data) and configures the underlying network without involving human resources. We call such a feature **Plug&Configure** and compare the existing approaches for proprietary real-time Ethernet protocols and TSN.

Despite existing concepts of applying logic-programming for modeling computer networks (declarative networking), there is no existing solution (except [132, 133] which are parts of this thesis) to build a network knowledge base covering data-centric application layer and TSN details. Time-triggered schedule synthesis for Ethernet and non-Ethernet networks is a well-known and discussed the problem. However, the presented solutions (except the synthesis tool Slate [101] which is expected to be developed in 2018) deal only with the theoretical fundamentals and do not address the complex engineering challenges which are required to generate and solve the constraints to compute IEEE 802.1Qbv-compatible schedules. In contrast, we propose a model-based constraint-solving system providing a graphical modeling tool to fully automate the synthesis procedure.

Evaluation of the related work shows that there are fewer contributions dealing with the analysis of schedule infeasibility in time-triggered networks such as TTEthernet and IEEE 802.1Qbv (the infeasibility analysis is also discussed in [46, 47] without providing high-level model correction assistance). Such an analysis is required to infer the origins of the design problems and correct them. Our contribution in [133] is further extended in this thesis. Formal timing analysis and simulation of TSN, AVB, and TTEthernet networks are well-discussed in the literature and are out of the scope of this thesis. The presented formal timing analysis approaches can use the modeling framework presented in this thesis as input and compute the worst-case latency and jitter of network streams. Verification of infrastructural requirements to support IEEE 802.1Qbv is a significant step before starting the synthesis. However, it has not been addressed in the literature (to the best of our knowledge).

The evaluation also shows that there are autoconfiguration proposals for industrial Ethernet (cf. [86]) which are helpful to understand the Plug&Configure challenges in TSN. Considering the existing industrial Ethernet technologies, the real-time communication control mechanism is part of the protocol itself and is coupled with the application layer. In contrast, TSN network components are clearly separated from the application layer. New concepts are required to enable TSN autoconfiguration (especially IEEE 802.1Qbv) at run-time which maintain architecture modularity and are independent of any specific application layer.

Self-configuration of TSN is also discussed in [96, 97, 143] where an agent-based approach is presented to learn the traffic and extract the required configuration parameters (with a similar motivation to our previously published contribution in [142]). In this thesis, concepts of [142] are further extended to incorporate TSN standards (focus on IEEE 802.1Qbv) in data-centric environments.

Table 2.1: Overview of related work and provided features.
● : Feature provided, ◐ : Feature partially provided (considerable adaptation work is required),
− : Feature not provided

References		Challenges										
		Network Modeling & Configuration				Model Verification & Correction (TSN/AVB/TTEthernet)					Plug & Configure	
Author	Year	Graphical Data-centric Network Modeler	Network Knowledge Base	Model-based Generated TT Constraints	TT Schedule Synthesis	Infeasibiliy Analysis	Correction Assistance	Formal Timing Analysis	Simulation	Infrastructural Verification of IEEE 802.1Qbv	Real-time Ethernet	IEEE 802.1Qbv
Mahfuzi et al. [37]	2018	-	-	-	●	-	-	-	-	-	-	-
Slate [101]		◐	-	●	●	-	-	-	-	-	-	-
Ashjaei et al. [75]	2017	-	-	-	-	-	-	●	-	-	-	-
Dvořák et al. [26]		-	-	-	●	-	-	-	-	-	-	-

References		Challenges										
		Network Modeling & Configuration				Model Verification & Correction (TSN/AVB/TTEthernet)					Plug & Configure	
Author	Year	Graphical Data-centric Network Modeler	Network Knowledge Base	Model-based Generated TT Constraints	TT Schedule Synthesis	Infeasibiliy Analysis	Correction Assistance	Formal Timing Analysis	Simulation	Infrastructural Verification of IEEE 802.1Qbv	Real-time Ethernet	IEEE 802.1Qbv
Hashemi et al. [133]		●	●	●	●	●	-	-	-	●	-	-
Maxim and Song [73]		-	-	-	-	-	-	●	-	-	-	-
Nayak et al. [38]		-	-	-	●	-	-	-	-	-	-	-
Pérez et al. [33]		-	-	-	●	-	-	-	-	-	-	-
Shi et al. [32]		-	-	-	●	-	-	-	-	-	-	-
Smirnov et al. [72]		-	-	-	-	-	-	●	-	-	-	-
Smirnov et al. [35]	2017	-	-	-	●	-	-	-	-	-	-	-
Stateflow [111]		◐	-	-	-	-	-	-	-	-	-	-
SysML [112]		◐	-	-	-	-	-	-	-	-	-	-
Wandeler and Thiele [114]		-	-	-	-	-	-	◐	-	-	-	-
Automation Studio [99]		◐	-	-	-	-	-	-	-	-	-	-
HiVision [100]		◐	-	-	-	-	-	-	-	-	-	-
SymTA/S [144]		◐	-	-	-	-	-	●	-	-	-	-
TwinCAT [98]		◐	-	-	-	-	-	-	-	-	-	-
Cao et al. [64]		-	-	-	-	-	-	●	-	-	-	-
Craciunas et al. [30]		-	-	-	●	-	-	-	-	-	-	-
Craciunas and Oliver [29]		-	-	-	●	-	-	-	-	-	-	-
Hashemi et al. [132]		-	●	-	-	-	-	-	-	●	-	-
Hashemi and Knoll [142]		◐	●	-	-	-	-	-	-	◐	-	●
Heise et al. [56]		◐	-	-	-	-	-	-	●	-	-	-
Heise et al. [94]		-	-	-	-	-	-	-	-	-	●	-
Laursen et al. [34]	2016	-	-	-	-	-	-	●	-	-	-	-
Pop et al. [36]		-	-	-	●	-	-	-	-	-	-	-
Saeedloei and Gupta [124]		-	◐	-	-	-	-	◐	-	-	-	-
Sagstetter [47]		-	-	-	●	●	-	-	-	-	◐	-
Schöler et al. [31]		-	-	-	●	-	-	-	-	-	-	-
Stähle [78]		◐	-	-	-	-	-	◐	-	-	◐	-
Steiner et al. [97]		-	-	-	-	-	-	-	-	-	-	◐
Thiele and Ernst [70, 71]		-	-	-	-	-	-	●	-	-	-	-
Dürkop et al. [93]		-	-	-	-	-	-	-	-	-	●	-
Gutiérrez et al. [96]		-	-	-	-	-	-	-	-	-	-	◐
Hammerstingl et al. [91]		-	-	-	-	-	-	-	-	-	●	-
Ko et al. [55]		◐	-	-	-	-	-	-	●	-	-	-
Martinez et al. [141]	2015	◐	-	-	-	-	-	-	-	-	◐	-
Regulin et al. [92]		-	-	-	-	-	-	-	-	-	●	-
Steinbach et al. [54]		◐	-	-	-	-	-	-	●	-	-	-
Tamasselicean et al. [66]		-	-	-	-	-	-	●	-	-	-	-
Thangamuthu et al. [68]		-	-	-	-	-	-	●	-	-	-	-
Thiele et al. [69]		-	-	-	-	-	-	●	-	-	-	-

References		Challenges										
		Network Modeling & Configuration				Model Verification & Correction (TSN/AVB/TTEthernet)					Plug & Configure	
Author	Year	Graphical Data-centric Network Modeler	Network Knowledge Base	Model-based Generated TT Constraints	TT Schedule Synthesis	Infeasibiliy Analysis	Correction Assistance	Formal Timing Analysis	Simulation	Infrastructural Verification of IEEE 802.1Qbv	Real-time Ethernet	IEEE 802.1Qbv
Axer et al. [63]	2014	-	-	-	-	-	-	●	-	-	-	-
Bordoloi et al. [62]		-	-	-	-	-	-	●	-	-	-	-
Boudjadar et al. [107]		◐	-	-	-	-	-	◐	-	-	-	-
Buckl et al. [90]		●	-	-	-	-	-	-	-	-	◐	-
Sagstetter et al. [46]		-	-	-	●	●	-	-	-	-	◐	-
Zhang et al. [27]		-	-	-	●	-	-	-	-	-	-	-
Zhao et al. [65]		-	-	-	-	-	-	●	-	-	-	-
Keddis et al. [89]	2013	-	-	-	-	-	-	-	-	-	◐	-
Dürkop et al. [87]		-	-	-	-	-	-	-	-	-	●	-
Hashemi et al. [116]		◐	-	-	-	-	-	-	-	-	-	-
Meyer et al. [53]		◐	-	-	-	-	-	●	●	-	-	-
Reimann et al. [60]		-	-	-	-	-	-	●	-	-	-	-
Alderisi et al. [51]	2012	◐	-	-	-	-	-	-	●	-	-	-
Diemer et al. [57]		-	-	-	-	-	-	●	-	-	-	-
Diemer et al. [79]		-	-	-	-	-	-	●	-	-	-	-
Dürkop et al. [86]		-	-	-	-	-	-	-	-	-	●	-
Lim et al. [50]		◐	-	-	-	-	-	-	●	-	-	-
Queck [59]		-	-	-	-	-	-	●	-	-	-	-
Steinbach et al. [52]		◐	-	-	-	-	-	-	●	-	-	-
Lim et al. [76]	2011	◐	-	-	-	-	-	-	●	-	-	-
Phan et al. [120]		◐	-	-	-	-	-	◐	-	-	-	-
Steinbach et al. [49]		◐	-	-	-	-	-	-	●	-	-	-
Steiner [25]		-	-	-	●	-	-	●	-	-	-	-
Vogel-Heuser et al. [117]		◐	-	-	-	-	-	-	-	-	-	-
Reinhart et al. [85]	2010	-	-	-	-	-	-	-	-	-	●	-
Steiner [28]		-	-	-	●	-	-	-	-	-	-	-
Hanzálek et al. [42]	2009	-	-	-	●	-	-	-	-	-	-	-
Hinrichs et al. [130]		-	◐	-	-	-	-	-	-	-	-	-
Loo et al. [128]		-	◐	-	-	-	-	-	-	-	-	-
Wang et al. [129]		-	◐	-	-	-	-	-	-	-	-	-
Zeng et al. [44]		-	-	-	●	-	-	-	-	-	-	-
Pop et al. [67]	2008	-	-	-	-	-	-	●	-	-	-	-
Limal et al. [106]	2007	◐	-	-	-	-	-	◐	-	-	-	-
Schioler et al. [115]		◐	-	-	-	-	-	◐	-	-	-	-
Witsch et al. [105]	2006	◐	-	-	-	-	-	◐	-	-	-	-
Jammes and Smit [82]	2005	-	-	-	-	-	-	-	-	-	●	-
Loo et al. [126], [125]		-	◐	-	-	-	-	-	-	-	-	-
Metzner et al. [39]		-	-	-	●	-	-	-	-	-	-	-

References		Challenges										
		Network Modeling & Configuration				Model Verification & Correction (TSN/AVB/TTEthernet)					Plug & Configure	
Author	Year	Graphical Data-centric Network Modeler	Network Knowledge Base	Model-based Generated TT Constraints	TT Schedule Synthesis	Infeasibiliy Analysis	Correction Assistance	Formal Timing Analysis	Simulation	Infrastructural Verification of IEEE 802.1Qbv	Real-time Ethernet	IEEE 802.1Qbv
Pop et al. [41]	2004	-	-	-	●	-	-	-	-	-	-	-
Shin and Lee [121]	2003	◐	-	-	-	-	-	◐	-	-	-	-
Pedreiras et al. [83]	2002	-	-	-	-	-	-	-	-	-	●	-
Gomaa [108]	2001	◐	-	-	-	-	-	◐	-	-	-	-
Pop et al. [40]	1999	-	-	-	●	-	-	-	-	-	-	-
Gupta and Pontelli [122]	1997	-	◐	-	-	-	-	◐	-	-	-	-
Larsen et al. [104]		◐	-	-	-	-	-	◐	-	-	-	-
Huber et al. [102]	1996	◐	-	-	-	-	-	◐	◐	-	-	-
Buck et al. [110]	1994	◐	-	-	-	-	-	-	◐	-	-	-
Broy et al. [103]	1992	-	-	-	-	-	-	◐	◐	-	-	-

Chapter 3

Approach Overview

3.1 Framework Architecture

To address the derived challenges and develop the required features from Section 2.5, we define the following requirements which are crucial for developing the modeling framework.

- Support for easy network modeling (R1): to design network topologies, hardware components, application's dataflow, timing requirements, etc., graphical and textual modeling elements are required. Such features are used to model the network by drag&drop of modeling elements. To develop the framework, software libraries and environments are required which assist to overcome this challenge.

- Easy access to the elements of a created network model (R2): after a network model is created, features are required which allow simplified access to the modeling elements by adequate queries. The main objective here is to find a solution which reduces the need for exhaustive programming of routines (e. g. using imperative or object-oriented languages) for each specific query.

- Selective representation of model information (R3): a network information management feature is required which enables efficient extraction of desired information for configuration and verification tasks. The efficiency means here reducing the overhead of imperative routine programming by following a declarative approach (which allows shorter programs and avoids side effects because of referential transparency).

- Support for logical inference (R4): to use the available network information for verification of desired properties and also for describing topological or application relations in the

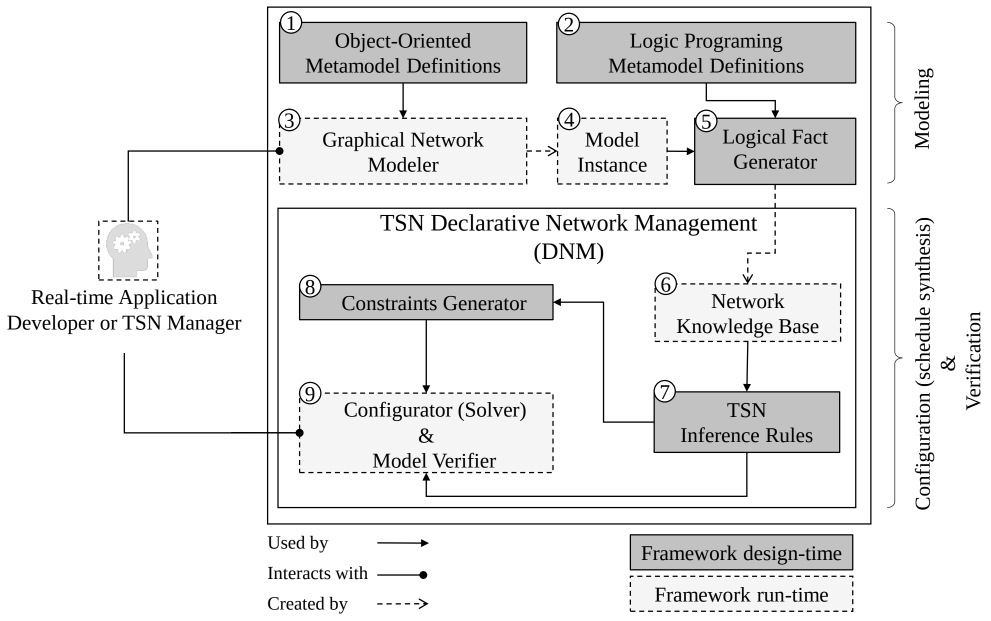

1 Object-Oriented Metamodel Definitions
2 Logic Programing Metamodel Definitions
3 Graphical Network Modeler
4 Model Instance
5 Logical Fact Generator
Modeling
TSN Declarative Network Management (DNM)
6 Network Knowledge Base
7 TSN Inference Rules
8 Constraints Generator
9 Configurator (Solver) & Model Verifier
Configuration (schedule synthesis) & Verification
Real-time Application Developer or TSN Manager
Used by
Interacts with
Created by
Framework design-time
Framework run-time

Figure 3.1: Overview of the modeling framework adapted from [133].

network, an inference engine is required which allows to derive logical proofs and to find desired unknowns (via queries).

- Support for constraint-based problem solving (R5): an environment is required where infrastructural and scheduling constraints can be formulated and solved such that the results can be used for configuration of network components (TSN switches).
- Support for constraint labeling (R6): the ability to understand design errors of network models and correcting them requires a feature that finds the conflicting constraints. Moreover, it has to allow labeling of constraints when they are created such that backtracing of the errors is possible.

As depicted in Figure 3.1, the modeling approach consists of an object-oriented metamodel (1) which builds the basis of a graphical network modeler (3) to created graphical model instances (4). Application developers or TSN managers use the graphical modeler (with drag&drop functionalities) to model network streams and annotate the QoS requirements of the developing real-time applications. For example, it can be annotated that a specific application contains periodic data transmission with tight timing constraints or it has reliability requirements and multiple disjoint paths (cf. *IEEE 802.1CB*) have to be provided. The modeling tool is also used to define the network topology and the physical properties of the components [133].

Using a formal metamodel based on logic programming (2) and following our previous work [132], the graphical model instances are transformed (5) into a network knowledge base (6). In this framework, Prolog is used as a concrete implementation of logic programming. Therefore, the graphical model instances are transformed into Prolog facts. Moreover, inference rules (7) are developed to exploit the power of logical inference which are required to

- query and verify network properties such as path redundancy (9), and
- to transform the required information from the network model into time-triggered schedule constraints (8),(9).

Using a Satisfiability Modulo Theories (SMT) solver, the constraints are solved and it will be decided whether there is a feasible schedule for the model. The constraints can also be used for verification of manually created schedules. The solver checks the schedule and decides if all constraints (real-time requirements) are fulfilled. These components are grouped and we call it TSN Declarative Network Management (DNM).

3.2 Object-oriented Metamodel

We develop a network metamodel to build the basis for the graphical modeler [133]. It consists of a *network view* which contains all graphical network components. A network *node* is either a *switch* or an *end-station*. Nodes consist of at least one Ethernet *port*. Switches and end-stations are connected through ports. A *link* connects two nodes and has exactly one *source* and one *target* port.

A real-time application is presented as a *domain* in the metamodel. Each domain contains one or more data *topics* which are *published* and *consumed* through ports within a domain. We distinguish between a *periodic topic* and a *sporadic topic*. The periodicity of the topics is relevant for time-triggered schedule synthesis. In contrast to periodic topics, the sporadic topics have an attribute *minInterval* for a minimum time between two consecutive frames. This parameter is important for worst-case latency analysis of a sporadic stream.

Data topics with critical timing constraints have to be modeled as periodic to be considered in the schedule synthesis constraints of IEEE 802.1Qbv. Each topic may have *QoS* requirements such as *timing* or *fail-operational*. For instance, a fail-operational QoS specifies the number of required redundant disjoint links. The object-oriented metamodel is presented in Figure 3.2.

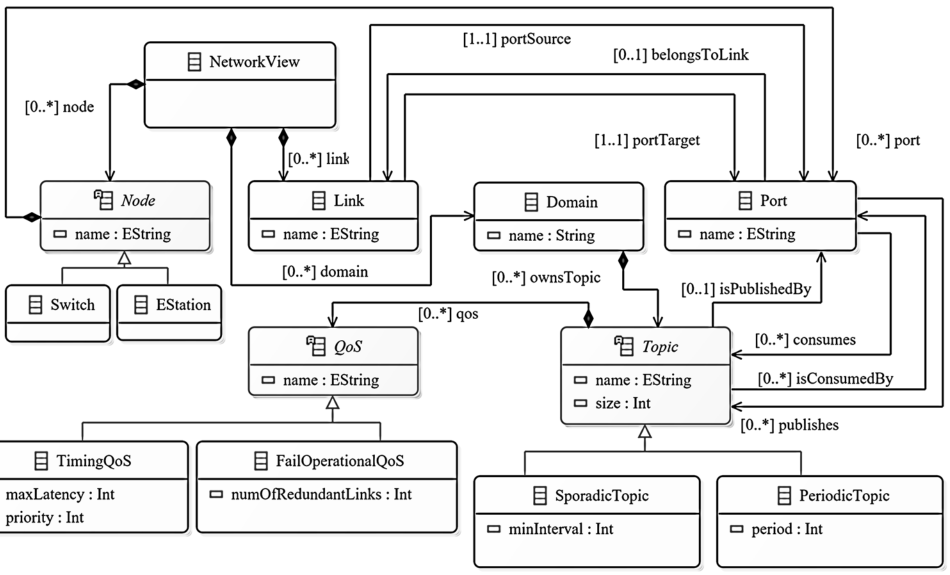


Figure 3.2: Object-oriented metamodel of the network.

3.3 Network Knowledge Base

To build a TSN knowledge base, Prolog (SWI implementation)[1] is used, which is a well-known logic programming language. The defined formal metalmodel of the knowledge base will follow the Prolog's syntax to define formal facts and rules. Prolog is based on Horn clauses and consists of *facts* and *inference rules*. Each rule has the form:

$$\alpha :- \beta_1, \beta_2, \ldots, \beta_n$$

which is equivalent to

$$\beta_1 \wedge \beta_2 \wedge \cdots \wedge \beta_n \implies \alpha$$

In Prolog, α is called *head* and $\beta_1, \beta_2, \ldots, \beta_n$ is the *body* of the rule. The head is true if the body is true. Each β in the body is a call to a *predicate*. The predicates in the body are known as *goals*. They can be either a fact (a clause with empty body) or a rule.

The knowledge base of a graphically modeled network consists of facts. Rules are defined to be able to describe logical (in the sense of mathematical logic) relations between facts and rules. We use rules for synthesis-related queries on the knowledge base in DNM. To present an introductory example, we take an automotive in-vehicle network. A simple fact as

[1]http://www.swi-prolog.org/

```
publishes(airbagSensorRight, airbagSensorRight_p1,
airbagDomain,airbagSensorRight).
```

describes that that the end-station *airbagSensorRight* uses its Ethernet port *airbagSensor-Right_p1* to publish a data topic *airbagSensorRight* in the *airbagDomain* domain. Accordingly,

```
firstPort(Domain,Topic,Port) :- publishes(_,Port,Domain,Topic).
```

is a rule that declares that *Port* is the first egress port to transmit *Topic* in *Domain*, if there is a fact *publishes* in the knowledge base with the desired values. The query to find the first egress port when publishing e. g. *airbagTrigger* topic is formulated as:

```
firstPort(airbagDomain,airbagTrigger,Port).
```

In this case, Prolog applies the availabe facts and inference rules to find a correct value for *Port* variable which is unknown. Variables always start with a capital. If there are matching options for *Port*, they are found and presented as query result. If not, the result is *false.*

In the following, the formal semantics of the facts are described (published in our previous work [133]). Moreover, we present the inference rules which have been developed to query the knowledge base regarding verification and schedule synthesis.

3.3.1 Network Facts

The modeling clauses consist of the following facts: *topic*, *qos*, *publishes*, *consumes*, *end-station*, *switch*, and *isLinked.*

Let NODE denote the set of all network *nodes* consisting of *switches* and *end-stations* denoted by the sets ESTATION and SWITCH, respectively: $\mathtt{NODE} = \mathtt{SWITCH} \cup \mathtt{ESTATION}$. Moreover, let PORT be the set of all Ethernet ports, which can be connected using links, where LINK denotes the set of all links. We denote with DOMAIN the set of all *domains* and with TOPIC the set of all *topics.*

For a better presentation, we also assume things to be named (labeled) with strings by an injective function.

$$\ell\colon (\mathtt{NODE} \cup \mathtt{LINK} \cup \mathtt{DOMAIN} \cup \mathtt{TOPIC}) \rightarrowtail \mathtt{STRING}$$

where STRING is the set of all possible finite words. Let $\mathcal{N} = (N, L, D)$ be a *network topology*, where $N \subseteq \mathtt{NODE}$ is the set of *nodes* of the network and $L \subseteq P \times P$ denotes the set of bi-directional *links* between node ports, where $P \subseteq \mathtt{PORT}$. Self-loops are prohibited, i. e., $\forall (p,p') \in L\colon p \neq p'$. Moreover, a port p can only be part of at most a single link, i. e., $|\{p \in P \mid (p,p') \in L \vee (p',p) \in L\}| \leq 1$.

Each node has a set of attached ports: ports: $\mathtt{NODE} \rightarrow \wp(\mathtt{PORT})$. It holds that each port is *unique*, i. e., $\forall n, n' \in N, n \neq n'\colon \text{ports}(n) \cap \text{ports}(n') = \emptyset$. Switches and end-stations use Ethernet ports to transmit or forward data topics embedded in Ethernet frames. Moreover, let $D \subseteq \mathtt{DOMAIN}$ be the set of all network *domains*. Each domain is assigned a set of *topics* $T \subseteq \mathtt{TOPIC}$: topics: $\mathtt{DOMAIN} \times \wp(\mathtt{TOPIC})$. Topics are *injectively* assigned to domains, i. e., $\forall d, d' \in D, d \neq d'\colon \text{topics}(d) \cap \text{topics}(d') = \emptyset$. Commonly used topic types are *periodic* (per) and *sporadic* (spo); the set of all topic types is referred to as $\mathtt{TTYPE} = \{\mathsf{per}, \mathsf{spo}\}$. Topic types are of importance for network configuration.

We derive *facts* from (i) the network topology (in particular nodes and links), (ii) topics and quality of service attributes, and (iii) published and consumed topics (wrt. data-centric middleware concept in Section 6.1). The derived facts are given next.

- A *switch fact* f^{s} is a pair (ν, P) where $\nu \in \mathtt{STRING}$ denotes its name and $P \subseteq \mathtt{PORT}$ is the set of the switch's ports. Analogously, an *end-station fact* f^{e} is defined as a tuple (ν, P) where $\nu \in \mathtt{STRING}$ denotes its name and $P \subseteq \mathtt{PORT}$ is the set of the end-station's ports.

- A *link fact* f^{l} is a triple (p, p', b) where $(p, p') \in L$ and $b \in \mathbb{N}_0^+$ is the link bandwidth.

- A *domain fact* f^{d} is a pair (ν, T) where $\nu \in \mathtt{STRING}$ denotes the name of domain d and $T = \text{topics}(d)$ is the set of owned topics.

- A *topic fact* f^{t} is a tuple $(\nu^{\mathsf{d}}, \nu^{\mathsf{t}}, \tau, \rho, \sigma)$ where $\nu^{\mathsf{d}}, \nu^{\mathsf{t}} \in \mathtt{STRING}$ denote the domain and topic name, respectively. The topic type is denoted by $\tau \in \mathtt{TTYPE}$ and its periodicity by $\rho \in \mathbb{N}_0^+$. Finally, $\sigma \in \mathbb{N}_0^+$ is the size of the topic.

- A *publishes fact* f^{p} and a *consumes fact* f^{c} are both tuples $(\nu^{\mathsf{n}}, \nu^{\mathsf{p}}, \nu^{\mathsf{d}}, \nu^{\mathsf{t}}) \in \mathtt{STRING}^4$ where $\nu^{\mathsf{n}}, \nu^{\mathsf{p}}, \nu^{\mathsf{d}}$, and ν^{t} denote the node, port, domain, and topic names.

- A *Timing QoS fact* f^{qt} and *Reliability QoS fact* f^{qr} are defined as tuples $f^{\mathsf{qt}} = (\nu^{\mathsf{q}}, \nu^{\mathsf{d}}, \nu^{\mathsf{t}}, \omega, \varrho)$ and $f^{\mathsf{qr}} = (\nu^{\mathsf{q}}, \nu^{\mathsf{d}}, \nu^{\mathsf{t}}, \zeta)$ where $\nu^{\mathsf{q}}, \nu^{\mathsf{d}}, \nu^{\mathsf{t}} \in \mathtt{STRING}, \omega, \varrho, \zeta \in \mathbb{N}_0^+$ denote the QoS name, domain name, topic name, deadline, priority, and number of redundant links. For periodic topics, we assume $\omega \geq \rho$. Following *IEEE 802.1Qbv* with 8 priority classes, one has $0 \leq \varrho \leq 7$.

```
streamFinder(Name,Domain,Topic,HelperList2,EgressPort,IngressPort
   ↪ ,Period,Length):-
    firstPort(Domain,Topic,FirstPort),
    lastPort(Domain,Topic,LastPort),
    path(FirstPort,LastPort,PortList),
    removeSwitchDevice(PortList,HelperList1),
    removeDevice(HelperList1,HelperList2),
    portClassifier(HelperList2,IngressPort,EgressPort),
    generateStreamName(Name,Topic,FirstPort,LastPort),
    topic(Domain,Topic,periodic,Period,Size).
```

Prolog 3.3: Inference rule to find periodic and critical streams caused by transmission of topics.

3.3.2 Network Rules

We developed a set of inference rules which relate the facts of the NKB and are used to extract information from NKB and to verify e. g. network reliability requirements. For instance, to make the *isLinked* fact symmetric, *link* rule is defined as:

```
    link(P1,P2,BW):- isLinked(P1,P2,BW).
    link(P1,P2,BW):- isLinked(P2,P1,BW).
```

Prolog 3.1: Rules for definition of bidirectional links.

To find end-stations, switches, and ports involved in transmission of a topic, *path* is defined:

```
    path(Start,End,Path) :-
    traveling(Start,End,[Start],Temp),
    reverse(Temp,Path).
```

Prolog 3.2: Rule to find nodes on transmission paths.

where *traveling* and *reverse* are helper predicates. The rule *streamFinder* finds all streams transporting a given *topic* and extracts all involved ingress and egress ports on the streams' paths. Considering IEEE 802.1Qbv, the egress ports are significant for schedule synthesis of periodic streams. It is important to find out, which periodicity ρ and data length σ a periodic stream has. These parameters are the basis for generating the stream schedule constraints. The *streamFinder* rule is presented in Prolog 3.3 where *removeSwitchDevice*, *removeDevice*, and *portClassifier* are helper predicates. It checks for a given topic, all communication paths

```
overlappingStreams(N1,D1,T1,P1,L1,N2,D2,T2,P2,L2,SharedEP):-
    streamFinder(N1,D1,T1,_,EP1,_,P1,L1),
    streamFinder(N2,D2,T2,_,EP2,_,P2,L2), N1\== N2,
    listIntersect(EP1,EP2,SharedEP).
```

Prolog 3.4: Rule to find which topics lead to overlapping streams.

```
allOverlappingStreams(T1,T2,Out) :-
    findall({"domain1":DStr1,"stream1":N1,"period1":P1,"length1":
       ↪ L1,"domain2":DStr2,"stream2":N2,"period2":P2,"length2":
       ↪ L2},(overlappingStreams(N1,D1,T1,P1,L1,N2,D2,T2,P2,L2,_
       ↪ ),atom_string(D1,DStr1),atom_string(D2,DStr2)),Out).
```

Prolog 3.5: Rule to find all overlapping streams.

from a publisher of the topic to its consumers including all ingress and egress ports on these paths. To each given pair of two topics, Prolog 3.4 is developed to find out streams with non-overlapping requirements that carry these topics where *listIntersect* is a developed helper rule to find the intersection of the egress port lists of each stream. Two streams have to be non-overlapping if they share at least one egress port. to find out non-overlapping streams carrying these topics where *listIntersect* is a developed helper rule to find the intersection of the egress port lists of each stream. Two streams have to be non-overlapping if they share at least one shared egress port. Using the rule *allOverlappingStreams* (Prolog 3.5), we collect all non-overlapping stream pairs in the network. These rules are used in Algorithm 2 to extracted all required input information to generate the constraints.

3.3.3 Model-based Generation of the TSN Knowledge Base

Using the defined logic programming metamodel, we transform the graphical model instance of a network into a knowledge base (Prolog facts). An excerpt of the transformation algorithm is demonstrated in Algorithm 1. The graphical model is analyzed to extract all periodic topics in the network dataflow which are used to build the developed Network Knowledge Base (NKB). The automated synthesis algorithm use NKB and inference rules to obtain information required for generating stream constraints.

The generated NKB is used for verification of desired network properties. Using adequate queries, network managers and application developers can, for example, verify that there are at least two disjoint paths between two given network nodes. Such a property is significant if there are fail-operational or redundancy requirements in the network. An example for such a use case is presented by [132].

The NKB can also be used as a network database which helps the network managers or application developers to obtain arbitrary information about the network architecture. Such information is helpful when e. g. new applications are developed or the network architecture has to be modified.

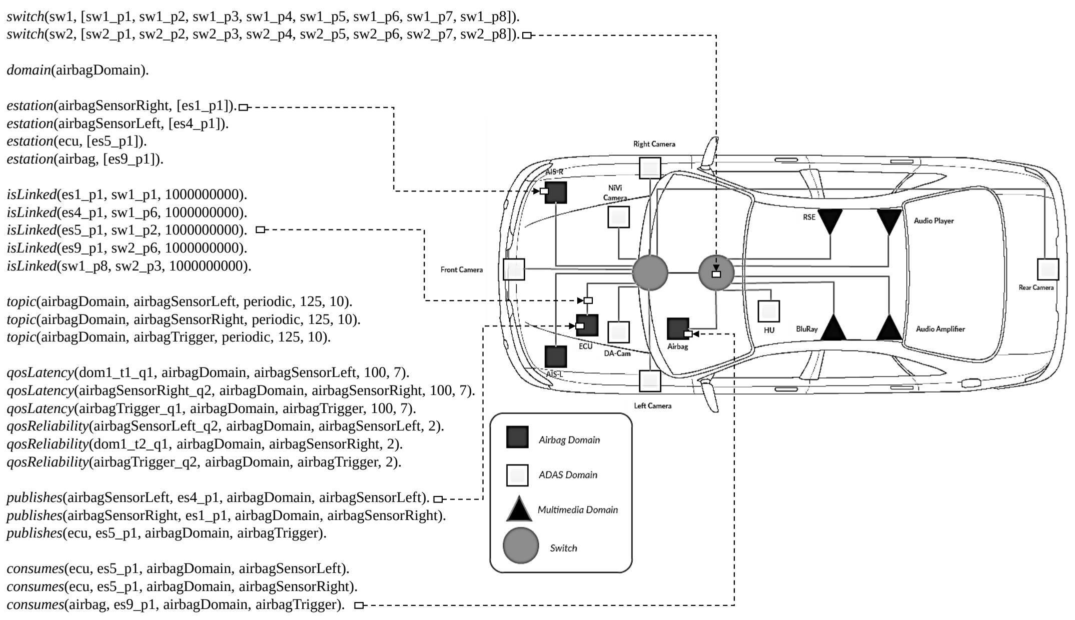
Algorithm 1 Adding periodic topics to the knowledge base

```
Input: $\mathcal{N} = (N, L, D)$                          ▷ Network derived from the model
Output: $NKB$                                              ▷ Network knowledge base
1: for all $d \in D$ do
2:     $\nu^{\mathsf{d}} \leftarrow d.getName()$
3:     for all $t \in \text{topics}(d)$ do
4:         if $periodic(t)$ then
5:             $\nu^{\mathsf{t}} \leftarrow t.getName(), \tau \leftarrow \mathsf{per}$
6:             $\rho \leftarrow t.getPeriod(), \sigma \leftarrow t.getSize()$
7:             $f^{\mathsf{t}} \leftarrow (\nu^{\mathsf{d}}, \nu^{\mathsf{t}}, \tau, \rho, \sigma).$
8:             $NKB.assertz(f^{\mathsf{t}})$                ▷ Add $f^{\mathsf{t}}$ to $NKB$
   return $NKB$                                            ▷ Returns the updated $NKB$
```

3.3.4 Exemplary Use Cases

In this section, a concrete example of an in-vehicle NKB is introduced from the automotive application domain. Using this example, different use cases are introduced where the presented logic-programming approach is very supportive. As a simple demonstration, an Advanced Driver Assistance System (ADAS) example from [145] is used. This example is further extended [132] to also consider tight real-time requirements for critical applications such as airbag. Two sensors are responsible for collecting collision information and an ECU, responsible for processing those data. The last node is an airbag trigger which will act according to the messages it gets from ECU. An excerpt of a simple NKB based on Prolog with focus on airbag domain is presented in Figure 3.3 and explained in the following.

The first *topic* fact is a predicate, describing that *airbagSensorLeft* is a *periodic* topic and belongs to *airbagDomain* domain with a period value of $125\mu s$ and a size of 10 Bytes. The first *qosLatency* fact (named *dom1_t1_q1*) indicates that there is a latency requirement on topic *airbagSensorLeft*. It has the VLAN priority of 7 and the maximum transmission latency is not



```
switch(sw1, [sw1_p1, sw1_p2, sw1_p3, sw1_p4, sw1_p5, sw1_p6, sw1_p7, sw1_p8]).
switch(sw2, [sw2_p1, sw2_p2, sw2_p3, sw2_p4, sw2_p5, sw2_p6, sw2_p7, sw2_p8]).

domain(airbagDomain).

estation(airbagSensorRight, [es1_p1]).
estation(airbagSensorLeft, [es4_p1]).
estation(ecu, [es5_p1]).
estation(airbag, [es9_p1]).

isLinked(es1_p1, sw1_p1, 1000000000).
isLinked(es4_p1, sw1_p6, 1000000000).
isLinked(es5_p1, sw1_p2, 1000000000).
isLinked(es9_p1, sw2_p6, 1000000000).
isLinked(sw1_p8, sw2_p3, 1000000000).

topic(airbagDomain, airbagSensorLeft, periodic, 125, 10).
topic(airbagDomain, airbagSensorRight, periodic, 125, 10).
topic(airbagDomain, airbagTrigger, periodic, 125, 10).

qosLatency(dom1_t1_q1, airbagDomain, airbagSensorLeft, 100, 7).
qosLatency(airbagSensorRight_q2, airbagDomain, airbagSensorRight, 100, 7).
qosLatency(airbagTrigger_q1, airbagDomain, airbagTrigger, 100, 7).
qosReliability(airbagSensorLeft_q2, airbagDomain, airbagSensorLeft, 2).
qosReliability(dom1_t2_q1, airbagDomain, airbagSensorRight, 2).
qosReliability(airbagTrigger_q2, airbagDomain, airbagTrigger, 2).

publishes(airbagSensorLeft, es4_p1, airbagDomain, airbagSensorLeft).
publishes(airbagSensorRight, es1_p1, airbagDomain, airbagSensorRight).
publishes(ecu, es5_p1, airbagDomain, airbagTrigger).

consumes(ecu, es5_p1, airbagDomain, airbagSensorLeft).
consumes(ecu, es5_p1, airbagDomain, airbagSensorRight).
consumes(airbag, es9_p1, airbagDomain, airbagTrigger).
```

Figure 3.3: An excerpt of the LP-based model of an in-vehicle network using the formally defined facts. Using inference rules, interesting properties of the network can be verified. The example is adapted from [145] and further developed in [132].

allowed to exceed $100\mu s$. Considering the first *publishes* fact, it declares that *airbagSensorLeft* end-station publishes the topic via its Ethernet port *es4_p1*. The published topic (*airbagSensorLeft*) is consumed by end-station *ecu*.

The predicate *switch* describes that e. g. *sw2* has eight ports $\{sw2\_p1, sw2\_p2, \ldots, sw2\_p8\}$. Similar to switch predicate, the *estation* facts are used to name the network end-station devices and their Ethernet ports which they provide for communication. To declare that there is a 1 Gigabit/s link between two switches, the Ethernet ports *sw1_p8* and *sw2_p3* are linked using the corresponding *isLinked* fact.

3.3.4.1 Network Configuration and Management

Using adequate queries, the configuration overhead for the network manager or application developer can be reduced. An imaginary example scenario: The end-station *airbagSensorRight* is panned to be integrated into the network considering the airbag application. Because of the highest criticality level, its data must be scheduled using time-aware shaper IEEE 802.1Qbv. The network manager requires to find out which egress ports are on the path of the new publishing device to its consumers. Such information is crucial e. g. to avoid a bottleneck in the

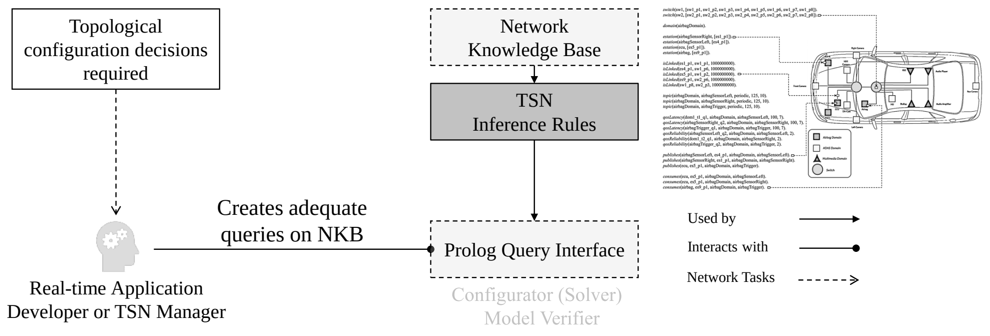

Topological configuration decisions required
Network Knowledge Base
TSN Inference Rules
Creates adequate queries on NKB
Prolog Query Interface
Real-time Application Developer or TSN Manager
Configurator (Solver) Model Verifier
Used by
Interacts with
Network Tasks

Figure 3.4: Using a Prolog query interface, required information for network configuration decisions are extracted from the network knowledge base.

```
% Prolog Query
:-streamFinder(_,airbagDomain,airbagTrigger,APorts,EPorts,_,_,_).
% Prolog Response
APorts = [es5_p1, sw1_p2, sw1_p8, sw2_p3, sw2_p6, es9_p1],
EPorts = [es5_p1, sw1_p8, sw2_p6];
```

Prolog 3.6: Query to find all ports and egress ports involved in transmission of *airbagTraigger* topic. Prolog responds with all ports and including all egress ports on the stream's path.

topology or to ensure that the affected egress ports support the required timing (features such as GCL and synchronization).

As the depicted in Figure 3.4, when new configuration tasks arrive, the network manager creates queries and interacts with the NKB using the Prolog query interface. Such an interface has access to the facts and rules of the network model and executes the Prolog inference engine where the queries can be processed. Using Prolog 3.6 query, streams which transport *airbagTrigger* topic are found. All involved Ethernet ports including egress ports are extracted which are on the path between publisher and consumer end-stations. These extracted ports must be considered when updating the GCL entries. With the help of the Prolog queries, complicated questions can be answered. For instance, the question of which data topics have the highest priority, go through switch *sw1*, and which period values do they have? With an appropriate Prolog query such as Prolog 3.7 the knowledge base is analyzed to find a solution.

3.3.4.2 Network Property Verification

Similar to the configuration use case, verification of the network properties can be supported by adequate queries. As depicted in Figure 3.5, safety experts use the Prolog query interface

```
% Prolog Query
:- topic(Domain,Topic,periodic,Period,_),
qosLatency(_,Domain,Topic,_,7), publishes(_,Port,Domain,Topic),
link(Port,SwitchPort,_), switch(sw1,PortList),
member(SwitchPort,PortList).
% Prolog Response
% First found topic
Topic = airbagSensorLeft, Period = 125,
Port = es4_p1, SwitchPort = sw1_p6;
% Second found topic
Topic = airbagSensorRight, Period = 125,
Port = es1_p1, SwitchPort = sw1_p1;
% Third found topic
Topic = airbagTrigger, Period = 125,
Port = es5_p1, SwitchPort = sw1_p2;
```

Prolog 3.7: An example for more complicated queries on NKB. Prolog responds with three found topics presenting the required information.

to verify the correctness of desired network properties. For this purpose, rules must be created only once based on the safety specification standards such as ISO 26262. Using available network facts and the created safety rules, verification queries are built. For instance, a requirement for critical applications is the support of a seamless redundancy mechanism such as IEEE 802.1CB. As an example, the airbag domain is highly critical and has fail-operational requirements. To verify that e. g. there are at least two disjoint paths between end-stations *ecu* and *airbagSensorLeft* a suitable query and the corresponding negative response are depicted in Prolog 3.8. Using this information, the safety expert sends a network design correction request to the designers or application developers. The model has to be modified such that the required property is fulfilled following an iterative approach. Other examples of infrastructural verification are presented in Section 4.4.

3.3.4.3 Time-triggered Schedule Synthesis

TSN time-triggered schedules are computed using specific application and network constraints such as the non-overlapping requirement of competing time-triggered streams. One challenging step to calculate a valid schedule is to build all relevant constraints which must be solved. It requires expert knowledge about the network and its application streams (including their

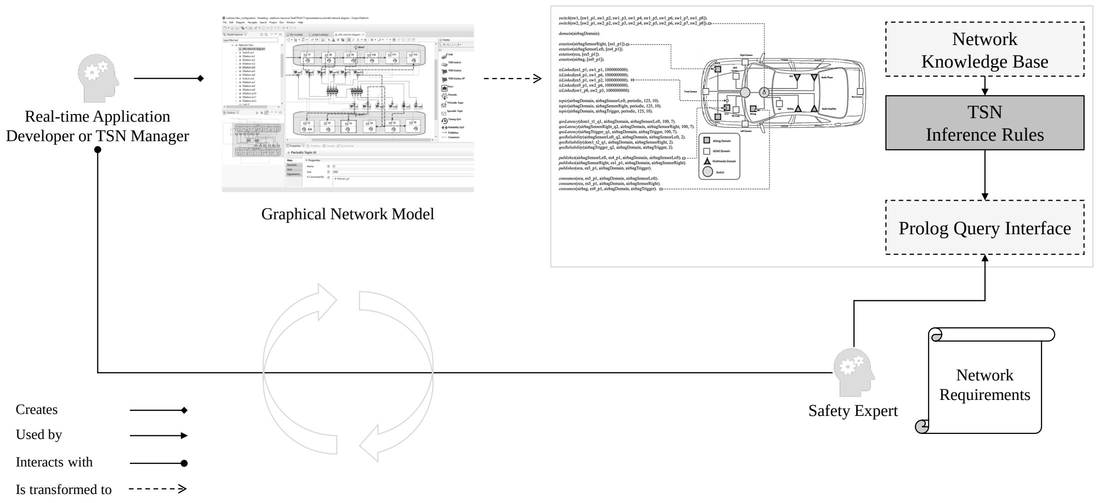


Figure 3.5: Using Prolog queries, network designers and safety experts work together to verify the required network properties to support real-time applications with e. g. fail-operational requirements iteratively.

```
% Prolog Query
:-findall(PL,path(es4_p1,es5_p1,PL),Z),length(Z,N),N>=2.
% Prolog Response
false;
```

Prolog 3.8: Verifiying of existing two physical disjoint paths between the left airbag sensor and the corresponding ECU.

timing requirements), details of TSN mechanisms such as GCL management, and expertise in a concrete constraint solving framework. Using Prolog queries, the required information from the network model is retrieved and automatically translated into constraints which are solved. This use case is further discussed in Section 4.3.

3.3.4.4 Model Optimization

The Prolog-based network knowledge base can be used to optimize the network model. For instance, long schedule synthesis for time-triggered streams can be avoided if expert knowledge is used and translated into Prolog rules. Such expert knowledge contains information about the fact that a highly divergent period value of streams which travel through the same egress ports increases the solving time. To avoid it, tradeoff values are defined as constraints (e. g. the standard deviation of streams' periods on an egress port is not allowed to be bigger than

1 *ms*). These constraints are verified by queries. This feature gets even more important when different engineers work simultaneously on different parts of the network model. In such scenarios, bottlenecks occur if streams travel between subnetworks (model segments). As an example, Figure 3.6 presents an in-vehicle network architecture following an domain-oriented approach (cf. [132,146]) where multiple network segments communicate over integration modules using TSN. Reducing synthesis time decreases the development time for architectures modifications.

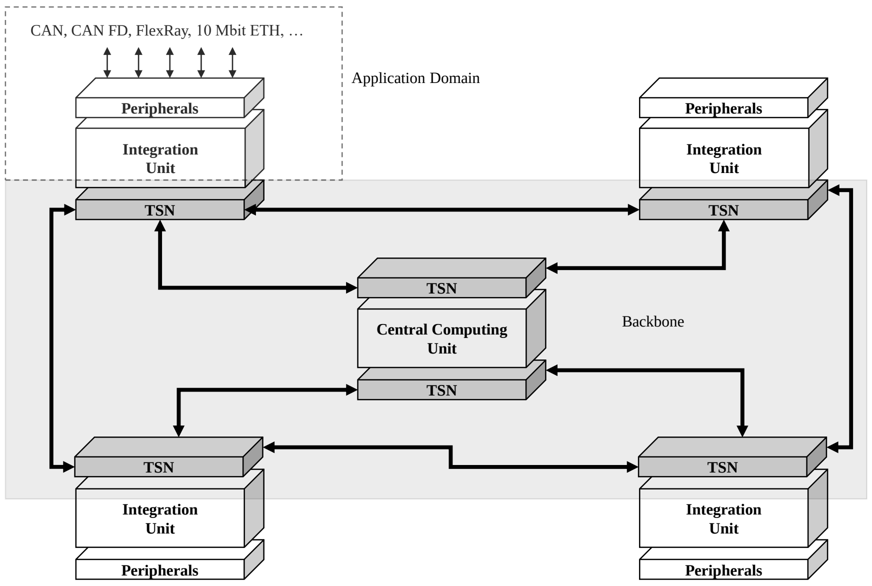


Figure 3.6: Minimizing of schedule synthesis time in cross-domain network architectures is required to accelerate the development process. Integration modules use the peripheral interfaces and communicate through a TSN backbone.

3.3.4.5 Plug-and-Play and Dynamic Reconfiguration

Dynamic configuration and verification for TSN networks at run-time are crucial to support features such as Plug&Produce. In a dynamic network, events occur at arbitrary times. For instance, if a new timing-critical actuator joins the network (e. g. a new robotic arm), new streams can come to the existence which can affect directly the current network schedule or bandwidth capacities of the network links. Using the Prolog-based knowledge base, events can be analyzed efficiently (in linear time because of Horn clauses [147]). Based on the analysis results (verifications), decisions can be made for further network management steps. This use case is the main topic of Chapter 6.

Chapter 4

Model-based Schedule Synthesis

4.1 Satisfiability Modulo Theories

Following the definitions in [148], in many mathematical problems, not all possible value assignments to variables are allowed. Using constraints, the legal combinations of value-to-variable assignments are specified. Let *scope* be a set of variables. A *tuple* on a scope S is defined as an assignment of a value to each variable in S. A *constraint* c is defined as a set of tuples on S. Constraints *involve* all variables in their scopes. For a set of variables S' where $S \subseteq S'$ and a tuple t on S', c satisfies t if $t \in c$ (restricted to S). For example, let $S' = \{x_1, x_2, x_3\}$ and $S = \{x_1, x_2\}$ where $x_1, x_2, x_3 \in \mathbb{N}$. A constraint c on S considering the variable order (x_1, x_2) is defined as:

$$c = \{t_1 = (9, 7), t_2 = (8, 8), t_3 = (8, 9), t_4 = (9, 9)\}$$

which simply describes $x_1 + x_2 \geq 16$. The constraint satisfies tuple $t_5 = (9, 7, x_3 = 5)$ because t assigns legal values to x_1, x_2.

A Constraint Satisfaction Problem (CSP) [148, 149] consists of (i) a set of variables, (ii) a value domain for each variable, and (iii) a set of constraints. To find a solution for a CSP, a value-to-variable assignment has to be found such that all constraints are satisfied. In this thesis, the schedule synthesis problem of TSN is formulated as a CSP using Satisfiability Modulo Theories (SMT). The variables which have the domain $\mathbb{N}$ express the time variable to open a port gate. Constraints express the timing constraints of the streams. The solution is used to configure TSN switches and end-stations. Satisfiability Modulo Theories (SMT) is a form of CSP and is about deciding the satisfiability of a first-order logic formula considering a background theory. Many applications do not require general first-order satisfiability, but rather satisfiability considering the underlying theory [150]. Using such theories, the interpretation of specific predicate and function symbols are fixed. For example, considering the following formula:

$$x > y + 2 \lor \neg(x < y)$$

applications using the linear integer arithmetic are not interested in satisfiability with nonstandard symbol interpretations but rather interested in deciding if the formula is satisfiable in an interpretation where $<$ is the usual ordering, and $+$ refers to addition function. Examples of theories of interests are *equality*, *uninterpreted functions*, *bitvectors*, and *arrays*. In this thesis, linear integer arithmetic is used as basic theory to solve scheduling constraints. The goal of a SMT solver is to find a *model* satisfying a given set of constraints or to prove that the constraints are unsatisfiable by returning an unsatisfiable core. SMT solvers find applications in e. g. software and hardware verification, planning, and network schedule synthesis. We use the model found by the solver to configure the network switches and end-stations. If no model is found, the returned unsatisfiable core is used to enhance the network design to find a model (cf. Chapter 5).

There is a set of various SMT solver implementations such as *Yices* [151] and *Z3* [152]. We use *Z3* to solve the schedule synthesis problem. As an introductory example, given the following constraints:

$$\begin{aligned}
C_1 :&(x \geq 0 \land y \geq 0),\\
C_2 :&y < 6,\\
C_3 :&x > 2,\\
C_4 :&y > x - 2,\\
C_5 :&\neg(y \geq -x + 8)
\end{aligned}$$

We aim to find a model that satisfy all constraints which are modeled in *Z3* as follows:

```
(declare-const x Int)
(declare-const y Int)
(assert (and (>= x 0)(>= y 0)))
(assert (< y 6))
(assert (> x 2))
(assert(> y (- x 2)))
(assert(not (>= y (+ (* x -1) 8))))
(check-sat)
(get-model)
```

The Z3 solves the constraints and assigns the integer value 2 to y and 3 to x producing the following output:

```
sat (model (define-fun y () Int 2) (define-fun x () Int 3) )
```

4.2 Constraints Overview

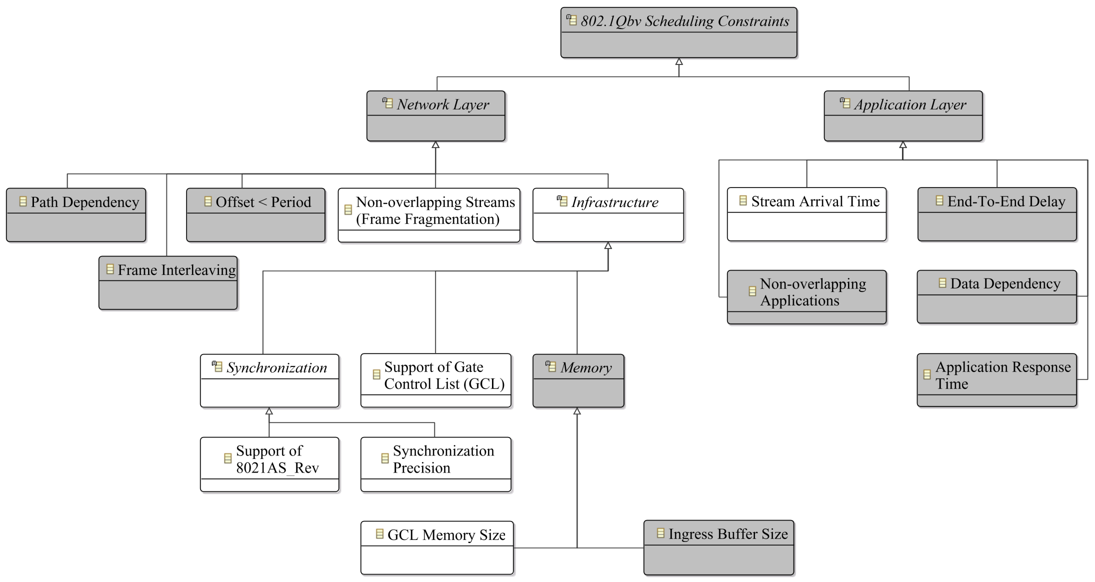


Figure 4.1: Constraint overview for TSN scheduling: Constraints in gray are discussed in the literature which we further extend in this thesis (white).

Considering the 802.1Qbv standard, there are multiple constraints which have to be considered. Following [27], these constraints can be divided into network and application constraints. The time-triggered constraints are defined by [28]. The non-overlapping constraints for streams ensure that there are no collisions of time-triggered reserved slots at egress ports on the path of the streams. This type of constraint is discussed in Section 4.3 and further extended to consider frame fragmentation. For larger streams than MTU, the fragmentation introduces more delay which must be considered in the synthesis procedure. Path-dependency constraints at network and application layers describe the following situation: considering a stream which travels through multiple network links, the scheduled time of this stream on each egress port has to be increased on the way from sender to receiver. In other words, a stream has to be first received before it can be sent out through an egress port. We use a more simplified approach in Algorithm 2 where gates remain open until a stream passes the last egress port. The offset of each time-triggered stream has to be less than its period. This constraint is defined by [30] as frame constraint. Similar to network constraints, application-related constraints for time-triggered communication are presented [27] who discuss data dependency and collision-freeness constraints in applications. In contrast to the existing application layer constraints, we introduce stream arrival time constraint which is crucial for synchronized actions in the network. It is discussed and presented in Section 4.4.3 with more details.

Before starting the generation of scheduling constraints, specific infrastructural properties and requirements must be verified. In this thesis, a set of infrastructural constraints are considered as presented in Section 4.4. They deal with synchronization ability, guaranteed precision, support of GCL mechanism, and GCL memory size constraints as discussed in Section 4.4.2.

To support modularity and further extensibility, the constraint types are modeled as hierarchical class diagrams. Following this approach new upcoming standards can be integrated into the framework (such as 802.1Qbu for frame preemption or 802.1CB to support redundancy).

4.3 Non-overlapping Constraints and Packet Fragmentation

Schedule constraints have to be generated which guarantee the fulfillment of hard real-time stream requirements. Non-overlapping constraints guarantee the latency and jitter requirements if IEEE 802.1Qbv mechanism is applied and the involved end-stations and switches are synchronized. Because these streams have the exclusive time-triggered link access, the interference with other traffic classes is avoided. Using the NKB, schedule synthesis constraints are generated. Generating these non-overlapping constraints are described in Algorithm 2.

The required network facts are obtained using adequate queries on KNB. Each topic f^{t} is always carried by a stream S^{t} from publisher $S^{\mathrm{t}}.pub$ to consumer $S^{\mathrm{t}}.con$ end-stations. Topics can be carried by multiple streams. The stream's variable for gate opening time $S^{\mathrm{t}}.tas$ (Line 10) is added to the solver which is required for the configuration and must be found by the solver. To calculate the transmission delay, the stream length is divided by the available bandwidth (Line 13).

In each network node, there is a processing delay which required to analyze the packet headers. The processing delay of each stream depends on the number of hops (Line 16). The interpacket gap delay of a specific stream is calculated using the number of MTU-sized frames of a stream multiplied with the defined interpacket gap α divided by the available bandwidth b (Line 17).

Algorithm 2 Periodic topics and non-overlapping constraints

```
1: $SMT$ : Solver
2: $\mathrm{lcm}(x, x')$    ▷ Least Common Multiple of $x, x'$
3: $\alpha$    ▷ Specified Minimum Interpacket Gap (8 Bytes for Gigabit Ethernet )
4: $MTU$    ▷ Maximum Transmission Unit
5: $\Phi_{proc}$    ▷ Switch processing delay
6: $\Phi_{prop}$    ▷ Propagation delay
7: $S^{\mathsf{t}}$    ▷ Stream carrying topic $f^{\mathsf{t}}$
8: for all $S^{\mathsf{t}} \in \mathcal{N} = (N, L, D)$ do
9:     $S^{\mathsf{t}}.tas$    ▷ Desired gate opening time variable of $S^{\mathsf{t}}$
10:    $SMT.assert(S^{\mathsf{t}}.tas)$    ▷ Add variable to the solver
11:    $b$    ▷ Available bandwidth
12:    $\Phi_{trans}$    ▷ Transmission delay
13:    $S^{\mathsf{t}}.\Phi_{trans} \leftarrow (\frac{S^{\mathsf{t}}.\sigma}{b})$
14:    $S^{\mathsf{t}}.pub, S^{\mathsf{t}}.con$    ▷ Publisher and consumer of $S^{\mathsf{t}}$
15:    $S^{\mathsf{t}}.hops$    ▷ # of hops from $S^{\mathsf{t}}.pub$ to $S^{\mathsf{t}}.con$
16:    $S^{\mathsf{t}}.\Phi_{proc} \leftarrow \Phi_{proc} S^{\mathsf{t}}.hops$
17:    $S^{\mathsf{t}}.\Phi_{ipg} \leftarrow \left(\left\lfloor \frac{S^{\mathsf{t}}.\sigma}{MTU} \right\rfloor + 1\right) \frac{\alpha}{b}$    ▷ Interpacket gap delay
18:    $S^{\mathsf{t}}.\Phi_{all} \leftarrow S^{\mathsf{t}}.\Phi_{trans} S^{\mathsf{t}}.hops + S^{\mathsf{t}}.\Phi_{proc} + S^{\mathsf{t}}.\Phi_{ipg} + \Phi_{prop}$
19:    $SMT.assert\,(S^{\mathsf{t}}.tas \geq 0 \wedge S^{\mathsf{t}}.tas \leq S^{\mathsf{t}}.\rho - S^{\mathsf{t}}.\Phi_{all})$
20: $NOS$    ▷ List of all non-overlapping stream pairs
21: for all $pair = \{S^{\mathsf{t}}_i, S^{\mathsf{t}}_j\}, pair \in NOS \wedge S^{\mathsf{t}}_i \neq S^{\mathsf{t}}_j$ do
22:    $\Psi_i \leftarrow \left[0, \frac{\mathrm{lcm}(S^{\mathsf{t}}_i.\rho, S^{\mathsf{t}}_j.\rho))}{S^{\mathsf{t}}_i.\rho} - 1\right]$
23:    $\Psi_j \leftarrow \left[0, \frac{\mathrm{lcm}(S^{\mathsf{t}}_i.\rho, S^{\mathsf{t}}_j.\rho))}{S^{\mathsf{t}}_j.\rho} - 1\right]$
24:    for all $\psi_i \in \Psi_i, \psi_j \in \Psi_j$ do
25:        $\gamma \equiv ((\psi_i S^{\mathsf{t}}_i.\rho + S^{\mathsf{t}}_i.tas + S^{\mathsf{t}}_i.\Phi_{all} <$
26:            $\psi_j S^{\mathsf{t}}_j.\rho + S^{\mathsf{t}}_j.tas) \vee$
27:            $(\psi_j S^{\mathsf{t}}_j.\rho + S^{\mathsf{t}}_j.tas + S^{\mathsf{t}}_j.\Phi_{all} <$
28:            $\psi_i S^{\mathsf{t}}_i.\rho + S^{\mathsf{t}}_i.tas))$
29:        $SMT.assert\,(\gamma)$
30: if $SMT.check() == SAT$ then    ▷ If constraints satisfied
31:    $\mathcal{C} = SMT.model()$    ▷ $TAS$ Configuration
32: else
33:    $\mathcal{U} = SMT.unsatCore()$    ▷ Conflicting constraints
```

In [27, 28, 30], the gate opening times of a periodic stream are different for each egress port on the path. In contrast, we sum all relevant delays (Line 18) and keep the gate open for the worst-case delay $S^{\mathsf{t}}.\Phi_{all}$. This way, the $S^{\mathsf{t}}.tas$ is the same for all egress ports on the path. It reduces the number of constraints but also reduces bandwidth utilization. The gate opening interval for a stream has to fit inside its period (Line 19).

We extract all non-overlapping stream pairs using *allOverlappingStreams* rule (Line 20). The non-overlapping constraints are generated and asserted (Line 29). When all constraints are generated and asserted to the solver context, the solving process starts (Line 30) to find a correct configuration (Line 31) or a proof that the constraints are not satisfiable (Line 33).

We can also use the automated synthesis procedure for timing verification of existing schedules. In this case, network managers or application evaluate use their manually (or using other tools) prepared schedule containing gate opening times for all streams. They use the modeling tool to generate schedule constrains and input the available gate opening times. The SMT solver tries to satisfy the constraints using the input data. If all constraints are satisfied, the manually generated gate opening times are verified. If the solver is not able to satisfy all constraint, it returns an unsatisfiable core indicating which stream constraints are unsatisfiable. We exploited this feature and demonstrate it in Chapter 5. As discussed in Algorithm 2, the overall delay of a stream $S^{\mathsf{t}}.\Phi_{all}$ is equal to:

$$S^{\mathsf{t}}.\Phi_{trans}S^{\mathsf{t}}.hops + S^{\mathsf{t}}.\Phi_{proc} + S^{\mathsf{t}}.\Phi_{ipg} + \Phi_{prop}$$

It corresponds to the window size which has to be exclusively reserved for $S^{\mathsf{t}}.\Phi_{all}$. However, $S^{\mathsf{t}}.\Phi_{trans}$ and $S^{\mathsf{t}}.\Phi_{ipg}$ require further refinements considering the following issues:

- For the calculation of the stream size $S^{\mathsf{t}}.\sigma$, packet headers of e. g. Ethernet, IP, UDP, etc. have to be considered too and not only the topic size σ.
- A topic f^{t} can travel through links providing different bandwidth b. It means the term $S^{\mathsf{t}}.hops$ has to be further refined.
- $S^{\mathsf{t}}.\Phi_{ipg}$ contains only the specified (according to IEEE 802.3) minimum interpacket gap delay. It does not consider the application or middleware specifications which could define larger $S^{\mathsf{t}}.\Phi_{ipg}$ values. For instance, a data-centric middleware could have a larger $S^{\mathsf{t}}.\Phi_{ipg}$ when e. g. frame fragmentation is required (this is the case if the topic size plus frame headers is greater than the network MTU size). Not considering this, leads to under provisioning for slot reservation which causes deadline misses.
- The MTU size can be different for each link.

Let $S^{\mathsf{t}}.L$ be the set of all physical Ethernet links involved in transmission of stream S^{t}. For each link $l \in S^{\mathsf{t}}.L$, we define l_{mtu} as the mtu size and l_b as the provided link bandwidth. To consider the transmission delays caused by packet headers, H is defined as the sum of the Ethernet header $H_{eth} = 18$ Bytes (includes also the VLAN header of 4 Bytes) and IP header $H_{ip} = 20$ Bytes, and the middleware or application specific headers defined as H_{app}:

$$H = H_{eth} + H_{ip} + H_{app}$$

The number of packet fragmentations on each link depends on its MTU size, the sum of packet headers, and the topic size of the stream (payload). A publisher has to send $H + S^{\mathrm{t}}.\sigma$ amount of data to a consumer. While H_{app} is sent only once, the number of fragmentations on a link is calculated as:

$$l_{fn} = \left\lceil \frac{H_{app} + S^{\mathrm{t}}.\sigma}{l_{mtu} - (H_{eth} + H_{ip})} \right\rceil$$

Using l_{fn}, the amount of data caused by fragmentations (packet headers) on a link is:

$$l_{frag} = (H_{eth} + H_{ip})\, l_{fn}$$

Based on the fragmentation overhead consideration of the packet headers, the transmission delay on each link is calculated using the respective bandwidth of the link:

$$S_l^{\mathrm{t}}.\Phi_{trans} = \left(\frac{H_{app} + S^{\mathrm{t}}.\sigma + l_{frag}}{l_b} \right)$$

The overall transmission delay of S^{t} is consequently equal to:

$$S^{\mathrm{t}}.\Phi_{trans} = \sum_{l \in S^{\mathrm{t}}.L} S_l^{\mathrm{t}}.\Phi_{trans}$$

Complementary to Algorithm 2, the interpacket gap delay $S^{\mathrm{t}}.\Phi_{ipg}$ from publisher to subscriber is computed based on the link bandwidth l_b of the publisher end-station and fragmentations number l_{fn} as:

$$S^{\mathrm{t}}.\Phi_{ipg} = max \left(\mathit{\Gamma}, \frac{\alpha}{l_b} \right) (l_{fn} - 1) + \frac{\alpha}{l_b}$$

where Γ presents the application or middleware specific interpacket gap delay. It depends on the application implementation and hardware capabilities of the publisher end-station. It deals with the question how a topic σ bigger than the allowed MTU is fragmented on the publisher side. More precisely, how long does the publisher delays the sending of topic fragments each after each other. We take the maximum (worst-case) value and multiply it with the number of fragmentations l_{fn}. After sending the last fragment, the standard interpacket delay ($\frac{\alpha}{l_b}$) is added which is required to avoid frame interference of different streams. As a summary, Line 18 of Algorithm 2 is updated (underlined terms) by:

$$S^{\mathsf{t}}.\Phi_{all} = \underline{S^{\mathsf{t}}.\Phi_{trans}} + S^{\mathsf{t}}.\Phi_{proc} + \underline{S^{\mathsf{t}}.\Phi_{ipg}} + \Phi_{prop}$$

4.4 Infrastructural Constraints

In addition to the defined basic constraints in [28] including the non-overlapping constraints, we consider infrastructural constraints which are directly related to the hardware capabilities of the switch ports and network topology required for IEEE 802.1Qbv.

4.4.1 Node Capability Constraints

Each device involved in transmission of time-triggered streams must support the synchronization approach defined in IEEE 802.1AS_Rev which is based on IEEE 1588 (PTP). Not only the synchronization ability is important but also the synchronization precision. For instance, if an end-station is on the path of a stream with a period of $S^{\mathsf{t}}.\rho = 100\mu s$, the required synchronization precision can be set for at least $100\mu s$. These can be verified using adequate facts and rules. Hence, the facts defined in Section 3.3.1 are extended as follows:

A *ptp fact* f^{ptp} is a tuple $(\nu^{\mathsf{n}}, \nu^{\mathsf{p}}, y)$ where $\nu^{\mathsf{n}}, \nu^{\mathsf{p}} \in \texttt{STRING}$ and $y \in \mathbb{N}_0^+$ denote node name, port name, and the synchronization precision in ns. More detailed and considering the PTP synchronization protocol, y is defined as the maximum offset value of a slave from its master which can be guaranteed. Considering Figure 3.3, exemplary implementation of this fact in Prolog is depicted in Prolog 4.1:

```
ptp(ecu,es5_p1,50).
ptp(airbagSensorLeft,es4_p1,10000).
ptp(airbagSensorRight,es1_p1,10000).
```

Prolog 4.1: Synchronization capability and precision of TSN nodes.

Using a new rule *verifyPTP* presented in Prolog 4.2, it can be verified if synchronitation requirements are fulfilled for a time-triggered stream carrying a specific topic (cf. Prolog 3.3):

```
verifyPTP(Domain,Topic,EgressPort):-
% Find streams and involved egress ports
% carying a topic of an application domain
streamFinder(Name,Domain,Topic,_,EgressPort,_,Period,_),
%for any port in egress ports on the stream's path
member(Port,EgressPort),
% ptp has to be suppoted and e.g. must be
% better than the stream's period
ptp(_,Port,Precision), Precision =< Period.
```

Prolog 4.2: Rule for verification of synchronization capabilities.

4.4.2 Port Programming Memory Constraints

Synchronization capability is not alone enough for transmission of IEEE 802.1Qbv streams. Each node involved in transmitting of such streams, has to support the GCL mechanism as explained in Section 1.1.2. To verify it before starting the synthesis, a new fact is defined. A *gcl fact* f^{gcl} is a tuple $(\nu^{\mathsf{n}}, \nu^{\mathsf{p}}, z)$ where $\nu^{\mathsf{n}}, \nu^{\mathsf{p}} \in \texttt{STRING}$ and $z \in \mathbb{N}_0^+$ denote the node name , port name, and the maximum allowed GCL entries which depends on the hardware memory size. In Prolog 4.3, the implementation of this fact is exemplary demonstrated for two different TSN switches.

```
gcl(sw1,sw1_p1,250).
gcl(sw2,sw2_p1,1000).
```

Prolog 4.3: Facts for verification of support of GCL mechanism and the maximum available GCL entries.

With an adequate rule presented in Prolog 4.4, it can be verified if all nodes (ports) on the path of a IEEE 802.1Qbv stream provide the GCL mechanism. The GCL configuration of the time-triggered slots has to be written in the memory of the relevant Ethernet port (not to be confused with port egress queue buffer). The amount of such a memory for each port has a significant impact on the overall switch cost. Also increasing such memory can lead to an increase in the hardware size which has disadvantages for space-limited application fields such as in-vehicle networks. Thus, this memory is limited and can store a defined maximum number of GCL entries.

To consider this limitation during the modeling phase, adequate constraints must be created.

```
verifyGCL(Domain,Topic,EgressPort):-
% Find streams and involved egress ports
% carrying a topic of an application domain
streamFinder(Name,Domain,Topic,_,EgressPort,_,_,_),
%for any port in egress ports on the stream's path
member(Port,EgressPort),
% gcl has to be supported and e.g. must
%provide e.g. at least 256 GCL entries
gcl(_,Port,Z), Z >= 256.
```

Prolog 4.4: Rule for verification of GCL capabilities.

For each Ethernet port in the model, its competing time-triggered streams have to be queried by the NKB. Let hp be the hyper period of all time-triggered (cf. Algorithm 3) streams LS^{t} to be sent out from an egress port ep. For all S^{t}, the sum of maximum numbers of the required time-triggered slots i. e. GCL entries is:

$$\Theta(ep) = \sum_{S^{\mathrm{t}} \in LS^{\mathrm{t}}} \left\lceil \frac{hp}{S^{\mathrm{t}}.\rho} \right\rceil \tag{4.1}$$

Each guard-band window also causes an entry in GCL and there is one gurad-band per time-triggered slot. We add the following constraint for each egress port to stay below the maximum number of allowed GCL entries:

$$2 * \Theta(ep) + \Theta'(ep) \leq z \tag{4.2}$$

where $\Theta'(ep)$ is defined as the maximum number of best effort slots. The value $\Theta'(ep)$ depends on the synthesized schedule: $0 \leq \Theta'(ep) \leq \Theta(ep)$. It can be reduced by trying to find a more compact time-triggered schedule where the remaining space between time-triggered slots are small enough to merge them and save extra GCL entries.

We note, that the infrastructural constraints can be verified either by NKB (using Prolog queries) or be formulating them as SMT constraints which are processed at schedule synthesis time. Using SMT to solve this kind of constraints together with other (e. g. non-overlapping) constraints could lead to inefficient network developing time, because of mixing a NP-complete problem (scheduling) with a problem (infrastructural verification) which can be solved efficiently by e. g. backward chaining approach (Prolog paradigm). However, if such a constraint is a very optimization objective for the schedule, it has to be formulated as a synthesis constraint.

Another reason to use Prolog approach is to increase the modularity of the framework. If a new (better) solver is supposed to be used, the NKB can remain unchanged. Only the constraint generator which uses NKB has to be modified to build specific constraints for different solver technologies.

4.4.3 Synchronized Actions

The topology plays an important role in considering the transmission delay of streams. Using the Store-and-Forward mechanism, the arrival times of synchronized streams will be different. Responsible factors for this are e. g. clock synchronization error, sending offset, and the number of hops from sender to receiver. For specific real-time applications which require synchronized physical actions such as motor and axis control applications, there is a maximum tolerable deviation in arrival times of synchronized streams. Figure 4.2 demonstrates an example application where the importance of synchronization errors is explained. Two step motors (*stepMotor_1* and *stepMotor_2*) have to be driven synchronously by *synchNode* using a periodic synchronization message. The line topology of the network consists of six switches. The motor end-stations are connected to two different switches. The synchronization messages are transmitted using two different streams. Ideally, the streams have to arrive at the same time, however considering the time-triggered paradigm, streams have different sending offsets to avoid overlapping. Moreover, one stream (to *stepMotor_1*) can reach its receiver sooner because of the line topology and the closer location of the synchronization node.

There is another control application which monitors the positions of the step motors and trigger an emergency stop if the physical behavior of the motors exceeds a tolerable safety-critical limit. To implement this application, step motor end-stations send periodically position values to the *emergencyStop* node. The control application compares the position values and stops them if needed by sending a stream including *stopMessage* to each motor. The synchronization errors in both directions could lead to safety-critical situations. Figure 4.3 depicts the different arrival times considering the exemplary GCL cycle time.

In contrast to [27, 28], where end-to-end latency constraints are considered in the schedule synthesis, we formulate the arrival constraints to cover the allowed deviations. The synthesis procedure considers these constraints and checks if there are conflicts. One possible approach is to define these as soft constraints. Following this approach, the solver tries to satisfy the soft constraints while solving standard (hard) constraints, however, it does not stop when there are

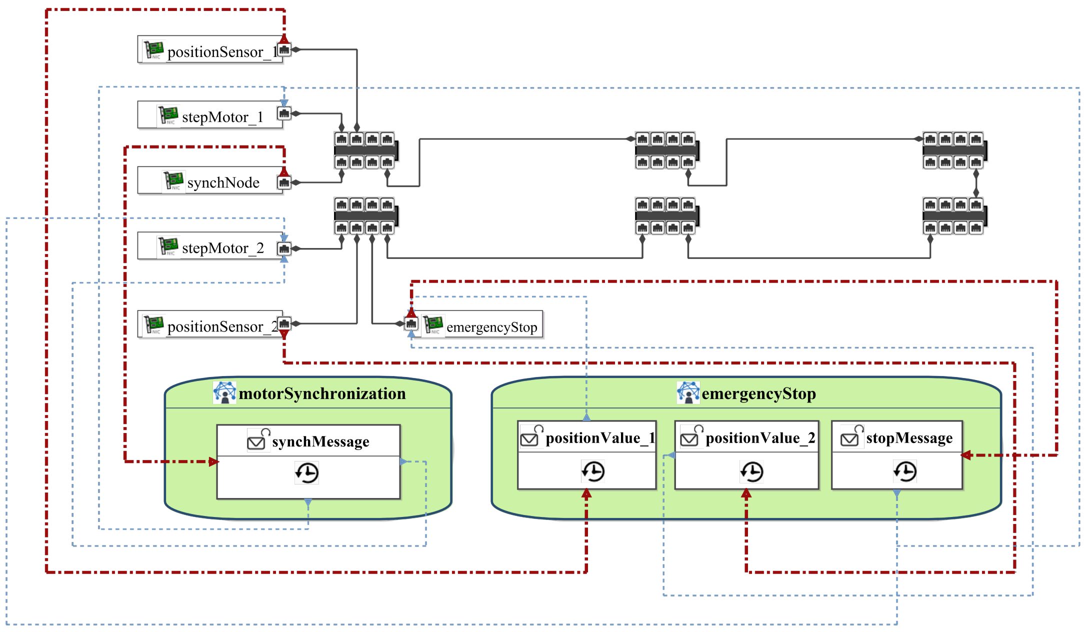


Figure 4.2: Store-and-Forward mechanism introduce extra transmission delay dependent on the topology and physical location of the end-stations. Variations in arrival times of synchronizing streams can have significant impact on synchronized actions (e. g. motor and axis control).

conflicting soft constraints. After the solving procedure ends, it can be checked which of these constraints are not satisfiable.

The developers should know about such a problem and have to find a solution to the application layer to implement synchronized actions successfully. For example, after receiving a stream, the start of the next cycle is calculated and the application code is executed directly when the next cycle starts (cf. Figure 4.3). Such a solution can be applied when waiting for the next cycle do not conflict with other application requirements. For instance, we consider a long line topology where one receiver is very close and the other one very far from the sender. In such a situation, the size of slots for time-triggered streams can be dramatically increased (because soft constraints are tolerated and do not stop synthesis) which leads to wasting of network bandwidth (which is required for bandwidth-intensive low-priority streams).

To avoid this, the arrival constraints can be formulated as hard constraints. In this case, the solver will stop and return conflicting constraints. Network designers have to analyze the labels of the conflicting streams and can further improve the network model. For example, the location of the far receiver can be modified and positioned closer to the sender. Let A^{sync} be a set of synchronized streams where $A^{sync}.e$ is the maximum allowed synchronization error

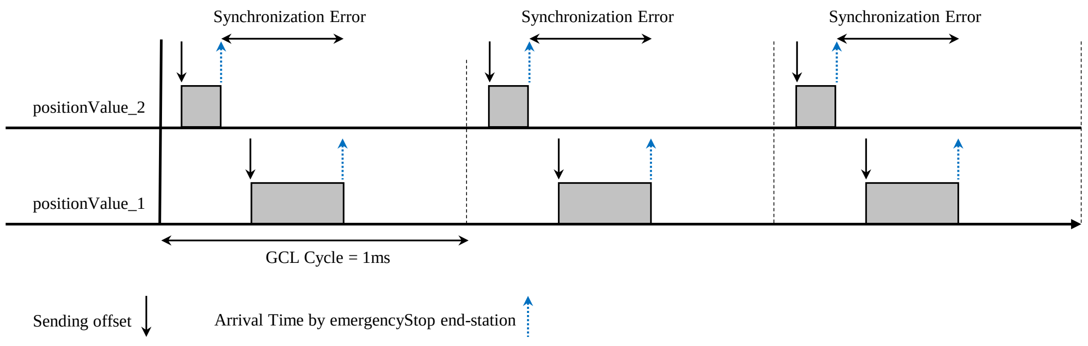


Figure 4.3: Different sending offsets and transmission delays based on the network topology cause synchronization errors. The maximum tolerable synchronization error can and must be formulated as scheduling constraints. The streams which transmit position values of the motors have the same size but position value of motor 1 later than position value of motor 2 (cf. Figure 4.2).

considering the arrival times. For each stream $S^{\mathrm{t}} \in A^{sync}$, we define its arrival time as $S^{\mathrm{t}}.a$:

$$S^{\mathrm{t}}.a = S^{\mathrm{t}}.tas + S^{\mathrm{t}}.\Phi_{all} \tag{4.3}$$

To guarantee that the maximum synchronization error is considered in the synthesis, the following constraint has to be added for each two arbitrary streams $S_i^{\mathrm{t}}, S_j^{\mathrm{t}} \in A^{sync}$:

$$|S_i^{\mathrm{t}}.a - S_j^{\mathrm{t}}.a| \leq A^{sync}.e \tag{4.4}$$

4.5 Preparing the Synthesis Result for 802.1Qbv

After all required verification queries are executed the synthesis can be started. The final result of the synthesis procedure is valid offset values for each IEEE 802.1Qbv stream provided the solver does not find any conflicting constraints or the synthesis terminates (NP-complete problem). Having the offsets of streams in a feasible schedule, Algorithm 3 can be used to transfer these to the relevant network nodes. The algorithm takes the list of all time-triggered streams of a specific port as input which is called LS^{t}. A list of streams periods L_p is defined that collects all periods to build the hyper period of the GCL for the given port hp. The GCL table consists of individual intervals i. Our objective is to compute these intervals and sort them by their starting time. For each interval its starting time i_{start} and duration $i_{duration}$ are computed based on Algorithm 2 and the resulting offsets ($S^{\mathrm{t}}.tas$). All computed intervals are appended to a list L_i and sorted based on i_{start} values. The sorted entries are the desired GCL entries with switching the gates for different priorities on and off. The GCL preparation approach is visually demonstrated in Figure 4.4.

Algorithm 3 Creating GCL entries for TSN ports (time-triggered priority is set to 7).

```
Input: $LS^{\mathrm{t}}$                                   ▷ List of all time-triggered streams of a specific port
Output: $GCL$
 1: $\mathrm{lcm}(x, x')$                                  ▷ Least Common Multiple of $x, x'$
 2: for all $S^{\mathrm{t}} \in LS^{\mathrm{t}}$ do
 3:     $L_p.append(S^{\mathrm{t}}.\rho)$                  ▷ List of all periods
 4:     $hp \leftarrow lcm(L_p)$
 5: $i$                                                    ▷ A time-triggered interval
 6: $L_i$                                                  ▷ list of intervals
 7: for all $S^{\mathrm{t}} \in LS^{\mathrm{t}}$ do
 8:     for $k \in \{0, \ldots, (\frac{hp}{S^{\mathrm{t}}.\rho}) - 1\}$ do
 9:         $i_{start} \leftarrow kS^{\mathrm{t}}.\rho + S^{\mathrm{t}}.tas$
10:         $i_{duration} \leftarrow S^{\mathrm{t}}.\Phi_{all}$
11:         $L_i.append(i)$
12: $S_i \leftarrow sort(L_i)$                             ▷ Sort the intervals by their start times
13: $e_i$                                                  ▷ GCL entry
14: $gb$                                                   ▷ guard band
15: for all $i \in S_i$ do
16:     $e_i \leftarrow (i_{start-gb}; 00000000)$
17:     $GCL.append(e_i)$                                  ▷ Append the new entry to GCL
18:     $e_i \leftarrow (i_{start}; 00000001)$
19:     $GCL.append(e_i)$
20:     $e_i \leftarrow (i_{start} + i_{duration}; 11111110)$   ▷ Opening gates for non-scheduled traffic
21:     $GCL.append(e_i)$
    return $GCL$
```

To communicate the computed GCL tables to relevant nodes, the standard IEEE 802.1Qcc is highly relevant where configuration approaches for TSN networks are discussed. According to this standard draft [20], a User/Network Interface (UNI) is specified where the user side represents stream Talkers and Listeners on the end-stations and the network side deals with switches in the network. It proposes distributed and centralized configuration approaches. The difference between two approaches is that the distributed one is designed to support credit-based shaper of AVB where dynamic stream reservation can be processed. Such a dynamic behavior is significant to reduce manual configuration where streams appear and disappear very often at the run-time. Considering automotive applications, infotainment streams are examples which require acceptable quality of service which can be handled by an adequate stream and bandwidth reservation.

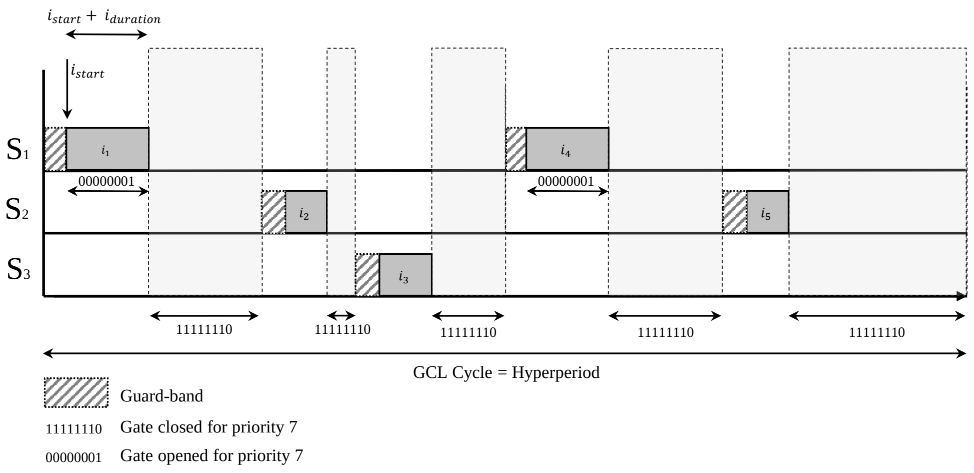


Figure 4.4: Using the computed offsets the GCL entries are created and communicated to the relevant TSN nodes for each affected port.

In contrast, the Centralized Network Configuration (CNC) is used to configure the following TSN features: (i) Credit-Based Shaper, (ii) Frame preemption, (iii) Time-triggered scheduled traffic, (iv) Frame replication and elimination for reliability, (v) Per-stream filtering and policing, and (vi) Cyclic queuing and forwarding. The draft mentions TSN use cases which require significant user configuration in the end-stations. For instance, in automotive applications and industrial automation with control applications, the timing requirements for TSN are derived from the I/O timing which depend on the physical environment under control e. g. closed loop sensor-actuator applications. The draft defines a Centralized User Configuration (CUC) entity to discover end-stations, capabilities, and requirements. The protocols that implement CUC are specific to user application and outside the scope of this draft. Our framework with its modeling and configuration features serves as a CUC/CNC implementation.

Moreover, the draft specified the YANG data modeling language to model configuration information in TSN networks. YANG is a remote network management protocol which exploits XML and JSON as encoding technologies. As future work, we plan to translate the output of our framework (computed schedules and GCL tables) in this modeling language. At the time of writing this thesis, this draft is still work in progress and the information mentioned here have to be checked with the current state of the document.

4.6 Time-Critical Sporadic Streams

In contrast to the periodic streams, time-sensitive event-based sporadic streams are sent at any given time. However, two successive streams must be separated by a specific and pre-defined time interval. The worst-case network load generated by a specific sporadic stream occurs when it sends with the highest frequency (the gap between sending times of two successive streams is minimum). In this case, such sporadic streams behave similarly to periodic streams. In this section, the focus is on sporadic streams with critical timing requirements. Representative applications for such streams are event-based critical user input devices (e. g. controlling a surgical device by surgeons) or alarming sensors with unknown triggering time.

In the previous section, the non-overlapping constraints of time-triggered streams with harmonic periods have been discussed. In this section, the integration of hard real-time requirements of both periodic and sporadic streams (defined by the deadlines) in the synthesis procedure is intended.

We assume that all sporadic streams will achieve their highest frequency. A sporadic stream sp_i is defined as a tuple $sp_i = (sp_i.\rho, sp_i.\sigma, sp_i.d)$ including the minimum time interval between two consecutive stream appearance, the size, and the deadline. To avoid unnecessary constraints building for the sporadic streams, we propose the following rapid test method to guarantee that all sporadic frames can hold their deadlines. If the test is successfully passed, there is no need to consider the sporadic streams in the synthesis process because it is formally verified that they do not exceed their deadlines independent from the time-triggered schedules on egress ports.

Considering the competing time-triggered streams LS^{t} , we use the notation of Algorithm 2. For each given sp_i, the worst-case delay caused by time-triggered streams $TW(sp_i)$ is calculated as follows:

$$TW(sp_i) = \sum_{S^{\mathrm{t}} \in LS^{\mathrm{t}}} \left(\left\lceil \frac{sp_i.d}{S^{\mathrm{t}}.\rho} \right\rceil (S^{\mathrm{t}}.\Phi_{all} + guardBand) \right) \tag{4.5}$$

where $\left\lceil \frac{sp_i.d}{S^{\mathrm{t}}.\rho} \right\rceil$ is the maximum possible number of time-triggered slots of S^{t} in $\Delta = [t, t + sp_i.d]$. Note that, all competing streams (sporadic and periodic) which have a shared egress port with sp_i are relevant.

Each sporadic stream can be delayed by maximal a lower priority frame (according to IEEE 802.1Q VLAN priority mechanism) at each hop from the sender to the receiver of sp_i. We calculate the maximum delay caused by theses as a result of multiplication of mtu and number of hops:

$$LW(sp_i) = mtu(sp_i.hops) \tag{4.6}$$

Adding all together, the worst-case delay of the low-priority and sporadic traffic is:

$$SW(sp_i) = \sum_{\substack{\forall sp_j \in SP \\ sp_j \neq sp_i}} \left(\left\lceil \frac{sp_i.\rho}{sp_j.\rho} \right\rceil sp_j.\Phi_{all} \right) + sp_i.\Phi_{all} + LW(sp_i) \tag{4.7}$$

Considering all competing sporadic streams SP, the minimum deadline, maximum deadline, and the maximum worst-case delay caused by sporadic streams are defined as:

$$D_{min} = sp_i.d \Leftrightarrow \forall sp_i, sp_j \in SP, sp_j \neq sp_i : sp_i.d \leq sp_j.d \tag{4.8}$$

$$D_{max} = sp_i.d \Leftrightarrow \forall sp_i, sp_j \in SP, sp_j \neq sp_i : sp_i.d \geq sp_j.d \tag{4.9}$$

$$SW_{max} = SW(sp_i) \Leftrightarrow \forall sp_i, sp_j \in SP, sp_j \neq sp_i : SW(sp_i) \geq SW(sp_j) \tag{4.10}$$

Similarly, the maximum worst-case delay caused by time-triggered streams can be calculated:

$$TW_{max}(sp_i) = \sum_{S^{\mathrm{t}} \in LS^{\mathrm{t}}} \left(\left\lceil \frac{D_{max}}{S^{\mathrm{t}}.\rho} \right\rceil (S^{\mathrm{t}}.\Phi_{all} + guardBand) \right) \tag{4.11}$$

Theorem 1. *No sporadic stream will exceed its deadline if* $(SW_{max} + TW_{max}) \leq D_{min}$.

Proof. Assume there is a sporadic stream sp_i that exceeds its deadline:

$$\exists sp_i : sp_i.d < SW(sp_i) + TW(sp_i) \tag{4.12}$$

$$sp_i.d < SW(sp_i) + TW(sp_i) \leq SW_{max} + TW_{max} \leq D_{min} \Rightarrow \tag{4.13}$$

$$sp_i.d < D_{min} \ \lightning \tag{4.14}$$

$\square$

The presented test is easy to implement and verify. However, it is significantly conservative depending on the number time-triggered and sporadic streams. Moreover If it fails, Backtracing of the problem is difficult because Theorem 1 considers all streams together and does not point to specific problematic streams. To overcome this issue, we propose the following test for each sporadic stream sp_i to find out which streams exactly can have a missed deadline in worst-case:

$$TW(sp_i) + SW(sp_i) \leq sp_i.d \tag{4.15}$$

The more complex the network, the bigger is the advantage of this method. In more complex network where many streams fulfill their timing constraints, designers are interested to find those small fractions which may have a timing problem in worst-case. To guarantee that sporadic streams do not miss their deadlines, we need to consider the sporadic streams and their deadlines in the synthesis procedure while shaping the time-triggered slots. In other words, it is required to generate constraints to consider and adapt the distance between time triggered slots such that there is enough time for periodic and sporadic streams to arrive before their deadlines.

To generate adequate constraints expressing the requirements of sporadic streams, a parameter is required, which describes the distance between time-triggered slots. For this purpose, we define α as minimum time between the end of a time-triggered slot and the start of the next consecutive time-triggered slot (similar concept to [25] where blank slots are introduced to transport rate constrained streams in the TTethernet protocol). We intend to calculate the correct α value such that the deadlines of all sporadic streams are respected. The number of α-slots in Δ is calculated as:

$$\sum_{S^{\mathsf{t}} \in LS^{\mathsf{t}}} \left(\left\lceil \frac{sp_i.d}{S^{\mathsf{t}}.\rho} \right\rceil \right) - 1 \tag{4.16}$$

Hence the time available for sporadic streams in $\Delta = [t, t + sp_i.d]$ is:

$$\alpha \left(\sum_{S^{\mathsf{t}} \in LS^{\mathsf{t}}} \left(\left\lceil \frac{sp_i.d}{S^{\mathsf{t}}.\rho} \right\rceil \right) - 1 \right) \tag{4.17}$$

We assume the worst-case order of sporadic streams which is formulated in Equation (4.7). Taking Equation (4.17) into account, we derive the following constraint for each sporadic stream sp_i:

$$\gamma' \equiv \alpha \left(\underbrace{\sum_{S^{\mathsf{t}} \in LS^{\mathsf{t}}} \left(\left\lceil \frac{sp_i.d}{S^{\mathsf{t}}.\rho} \right\rceil \right) - 1}_{N} \right) - SW(sp_i) \geq 0 \tag{4.18}$$

which means that the sum of α-slots has to be large enough for transmission of sp_i and all other sporadic streams (worst-case scenario).

Theorem 2. *If γ' is satisfied for a sporadic stream, its payload will be transmitted before its deadline.*

Proof. Assume γ' is satisfied for all streams but one stream sp_i misses its deadline. We will have two cases:

Case 1:

$$
\begin{gathered}
\exists sp_k : \alpha N - SW(sp_i) - sp_k.\Phi_{all} < 0 \Rightarrow \\
sp_k \text{ delays } sp_i \Rightarrow \\
sp_k \text{ arrived before } sp_i \\
sp_k \in SW(sp_i) \Rightarrow \\
SW(sp_i) \textit{was not the worst-case delays caused by sporadic streams} \lightning
\end{gathered}
$$

Case 2:

$$
\begin{gathered}
\exists I : I \text{ delays } sp_i, \text{ where } I \text{ is a not considered time-triggered slot} \Rightarrow \\
\alpha\,(N+1) - SW(sp_i) < 0 \Rightarrow \\
\alpha N + \alpha - SW(sp_i) < 0 \qquad (4.19) \\
\text{without } I : \\
\alpha N - SW(sp_i) \geq 0 \Rightarrow \\
-\alpha N + SW(sp_i) < 0 \qquad (4.20) \\
\text{Equations (4.19) and (4.20)} \Rightarrow \alpha < 0 \lightning
\end{gathered}
$$

□

To consider α in the schedule of the time-triggered periodic streams, γ from Algorithm 2 has to be reformulated as:

$$
\begin{gathered}
\gamma \equiv \big((\psi_i S^{\mathsf{t}}_i.\rho + S^{\mathsf{t}}_i.tas + S^{\mathsf{t}}_i.\Phi_{all} + \boldsymbol{\alpha} < \psi_j S^{\mathsf{t}}_j.\rho + S^{\mathsf{t}}_j.tas) \\
\vee \\
(\psi_j S^{\mathsf{t}}_j.\rho + S^{\mathsf{t}}_j.tas + S^{\mathsf{t}}_j.\Phi_{all} + \boldsymbol{\alpha} < \psi_i S^{\mathsf{t}}_i.\rho + S^{\mathsf{t}}_i.tas)\big)
\end{gathered}
$$

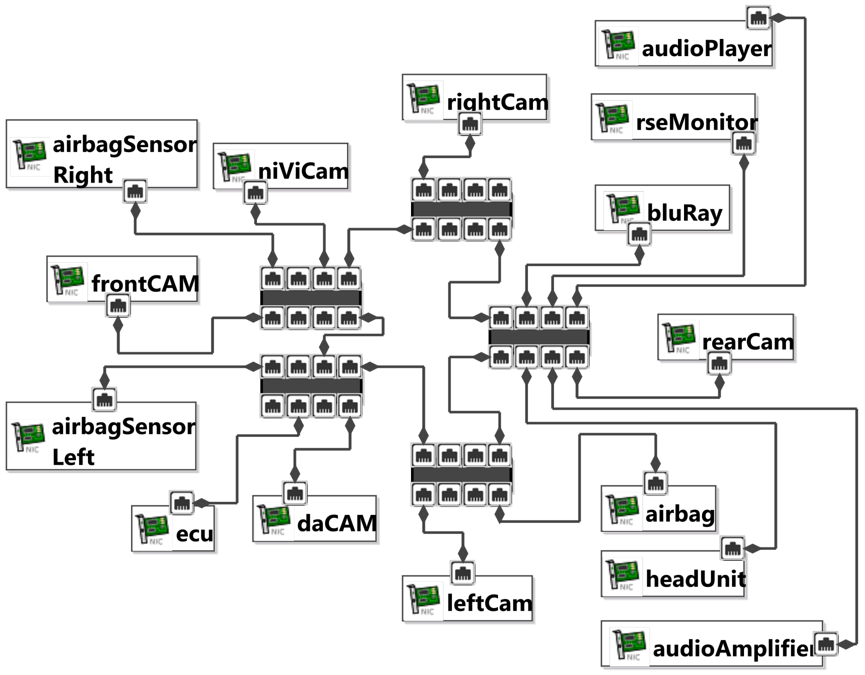


Figure 4.5: Modeling network topology using the graphical editor.

4.7 Implementation

In this section, the prototypical implementation of the framework components is discussed. Moreover, the synthesis results of an exemplary TSN network with two switches are presented.

4.7.1 Framework Components

Two roles (taken by a human) are defined to interact with the framework. Network designers who create and optimize graphical network models and safety experts who verify whether the model satisfies the infrastructural and safety requirements (e. g. related to IEEE 802.1Qbv and IEEE 802.1CB).

The graphical network modeler is implemented based on the Eclipse Modeling Framework (EMF)[1] using Java programming language. The object oriented-meta model of the modeler is discussed in Section 3.2. To demonstrate the modeling tool capabilities, a modified in-vehicle automotive network example from [132] is used. It consists of 5 TSN switches with each 8 ports. End-stations are connected to these switches and build together the hardware topology as demonstrated in Figure 4.5. To derive the topology facts correctly, infinite loops in the

[1]http://www.eclipse.org/modeling/emf/

```
path(Start,End,Path) :- pathHelper(Start,End,[Start],Temp),
reverse(Temp,Path).
pathHelper(X,Y,P,[Y|P]) :- link(X,Y,_).
pathHelper(X,Y,Visited,Path) :- \+member(Z,Visited),
Z \== Y, link(X,Z,_),
pathHelper(Z,Y,[Z|Visited],Path).
```

Prolog 4.5: Rule to consider infinite loops.

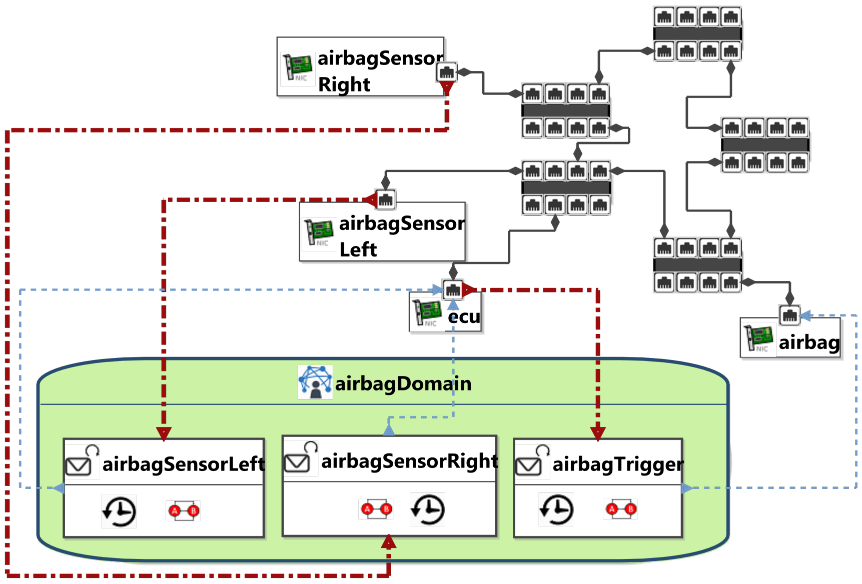


Figure 4.6: Modeling dataflow and QoS requirements using the graphical editor

topology has to be determined and considered. Such loops are caused by cycles in e. g. ring structures. A solution for this problem is implemented in Prolog 4.5 where the visited nodes will be remembered. The end-stations belong to different domains (applications). The cameras are e. g. used in driver assistance system domain, airbag sensors and actuators are involved in the real-time airbag domain, and multimedia end-stations belong to the entertainment domain. Having a closer look in the dataflow of the airbag domain as depicted in Figure 4.6, *airbagSensorRight* and *airbagSensorLeft* data topics are published by the sensors and by *ecu* end-station. It controls the sensors' data and publishes *airbagTrigger* topic which is consumed by airbag actuator. The QoS requirements are inside the topics. For instance, *airbagTrigger* topic is a periodic topic and has both timing and reliability requirements. The appropriate parameter values such as period and size, and etc. are entered textually in the modeling tool (not demonstrated here).

The presented graphical modeler also offers a feature to filter (hide) uninteresting visual information in the model. It is very useful for more complex network designs where multiple designers work on different segments of the network model. For instance, teams can work on different application domains and are not interested in the visual details of other domains. Without the filtering feature, the probability for design mistakes will be increased and slow down the network developments. Moreover, it is advantageous for model optimization (e. g. avoiding bottlenecks), when designers can use the selective visual information to reduce cross-domain communication (if not necessary) to increase the locality in the model and mitigate model complexity. An example for such a scenario is presented in Figure 3.6 where different automotive application developers focus on different parts of the network defined as integration units. Considering large networked production lines in factories and application of TSN networks in many automation domains, this modeling framework offers a scalable solution for modeling, configuration, and verification.

An overview of these components and roles and the temporal interactions between them is depicted in Figure 4.7. The created graphical model is sent to a Java-to-Prolog model transformer where adequate Prolog facts are created. These components are implemented in Java and traverse the graphical model to build the facts. Network designers can formulate optimization queries to check the existence of e. g. bottlenecks (competitive streams on a single egress port with high deviation in period length) to decrease synthesis time. These queries are sent to the model verifier where the defined optimization rules are used to check the trueness of the queries. It is an iterative procedure where the graphical model is modified and checked until the model satisfies the designer's desires. The optimized model is sent to the network safety expert for safety verification. For this purpose, a set of network safety rules are (already) created based on general application requirements such as redundant links (e. g. ISO 26262) or based on network properties which have to exist to support TSN standards. The safety expert formulates verification queries and lets them be checked by the model verifier. An example related to this verification step is given in Section 3.3.4.2.

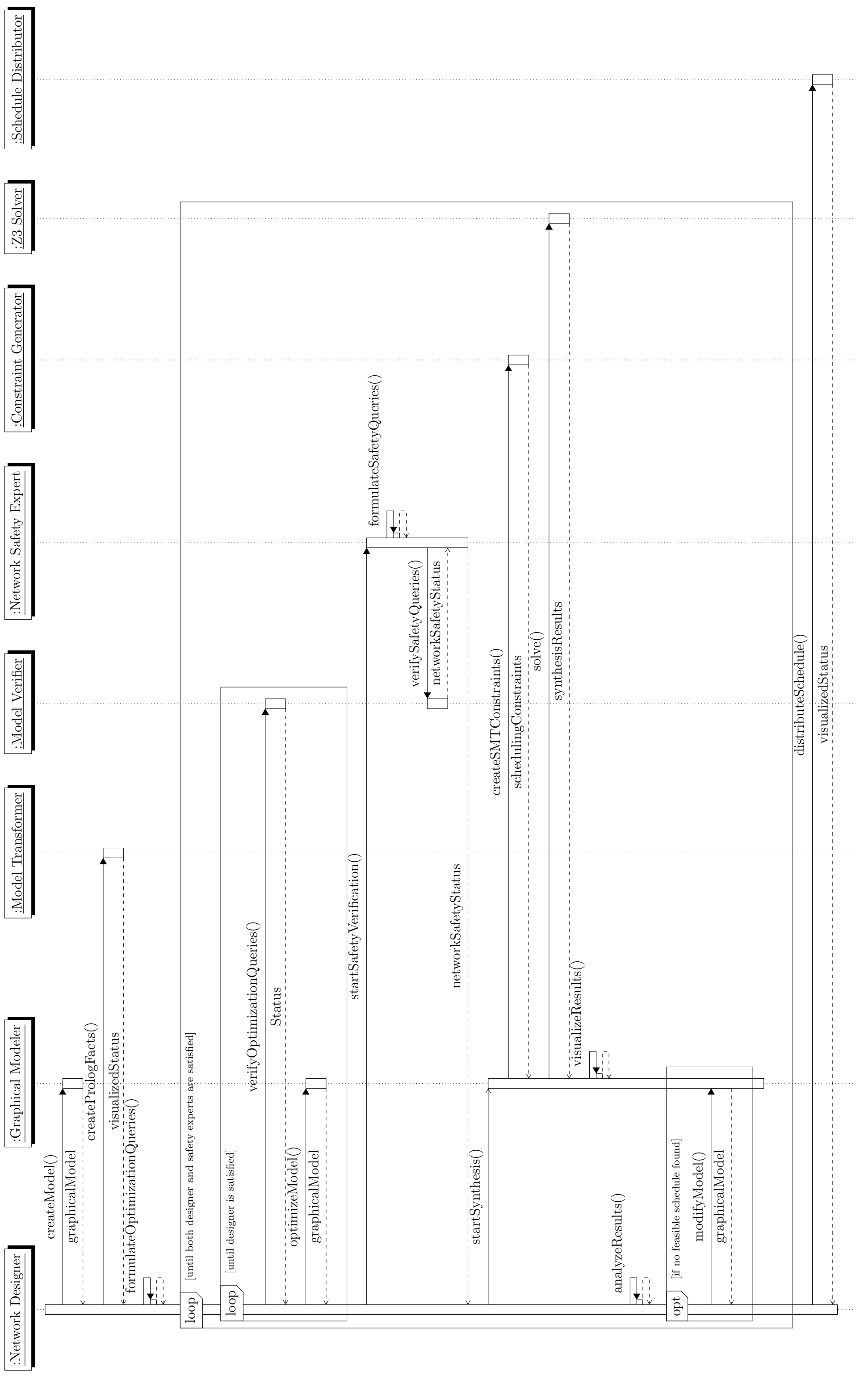
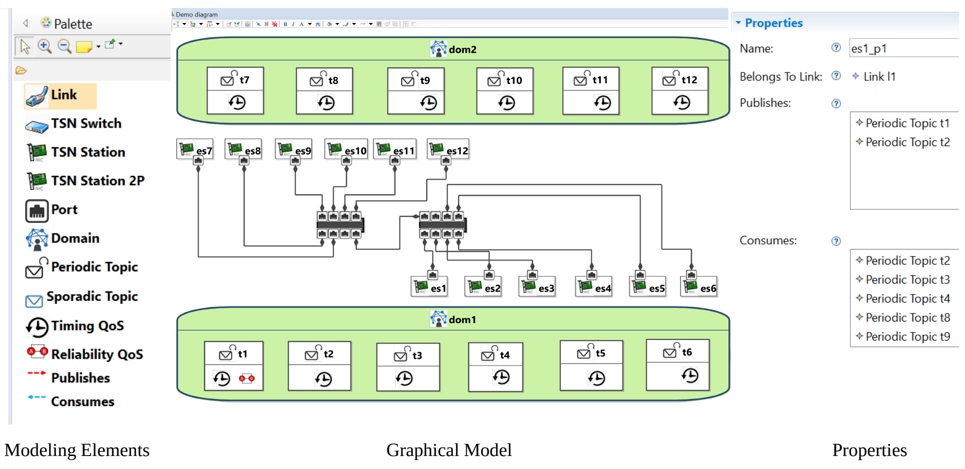


Figure 4.7: Sequence diagram of the framework from modeling to the distribution of the synthesized schedule.



Name: es1_p1
Belongs To Link: Link l1
Publishes: Periodic Topic t1, Periodic Topic t2
Consumes: Periodic Topic t2, Periodic Topic t3, Periodic Topic t4, Periodic Topic t8, Periodic Topic t9

Figure 4.8: An excerpt of the modeling elements, created graphical model, and node properties of the case study (published in [133]).

If the verification results are satisfying, the network designer starts the schedule synthesis by interacting with the graphical modeler. The modeler calls the constraint generator functionality which is Python script to build adequate constraints for a Z3 solver[1]. The solver module is also implemented in Python and use the Z3 interfaces. When the synthesis is finished, the results are visualized for the designer. There are two possible situations. If the model is not satisfiable (which means no feasible schedule has been found by the solver), the designer will analyze the labels of the conflicting constraints to find out the possible reasons for unsatisfiability. Iteratively, the model is modified by the designer until a feasible schedule is found. Exploiting unsatisfiability cores of the model is discussed in Chapter 5. However, if a feasible schedule is found, the computed offsets for the time-triggered streams are sent to a Python script which implements Algorithm 3. The schedule distributor calculates the GCL tables which are required for configuration. It executes shell commands to connect to nodes (e. g. TSN switches) and send them the GCL updates.

4.7.2 Demonstration

The objective of this demonstration is to show the capability of the graphical modeling tool to synthesize a feasible schedule based on IEEE 802.1Qbv. In Figure 4.8, an exemplary network

[1]https://github.com/Z3Prover/z3

Table 4.1: Stream properties and schedule synthesis results of the demonstration use case [133].

Stream	Non-overlapping with	Domain	Topic	Publisher	Consumer	Period (ns)	Length (Byte)	**First Gate Opening Time (ns)**
s_1	s_2	*dom1*	t_1	$es_1\_p_1$	$es_2\_p_1$	500.000	400	38.072
s_2	s_1	*dom1*	t_2	$es_1\_p_1$	$es_2\_p_1$	2.000.000	1000	0
s_3	s_4, s_5, s_6	*dom1*	t_3	$es_2\_p_1$	$es_1\_p_1$	8.000.000	3000	0
s_4	s_3	*dom1*	t_4	$es_2\_p_1$	$es_3\_p_1$	16.000.000	6000	8.103.096
s_5	s_3, s_7, s_8, s_6	*dom2*	t_7	$es_7\_p_1$	$es_1\_p_1$	500.000	400	186.480
s_6	s_{13}, s_3, s_5, s_{12}	*dom2*	t_{12}	$es_{11}\_p_1$	$es_1\_p_1$	100.000.000	5000	7.705.352
s_7	s_8, s_5	*dom2*	t_7	$es_7\_p_1$	$es_8\_p_1$	500.000	400	0
s_8	$s_5, s_7, s_9, s_{10}, s_{12}, s_{13}$	*dom2*	t_7	$es_7\_p_1$	$es_{12}\_p_1$	500.000	400	167.608
s_9	$s_8, s_{14}, s_{10}, s_{11}, s_{12}, s_{13}$	*dom2*	t_8	$es_8\_p_1$	$es_{12}\_p_1$	500.000	400	186.480
s_{10}	$s_8, s_9, s_{11}, s_{12}, s_{13}$	*dom2*	t_9	$es_9\_p_1$	$es_{12}\_p_1$	5.000.000	5000	0
s_{11}	$s_8, s_9, s_{10}, s_{12}, s_{13}$	*dom2*	t_{10}	$es_9\_p_1$	$es_{12}\_p_1$	100.000.000	10000	99.205.352
s_{12}	$s_{11}, s_{10}, s_9, s_8, s_{13}, s_6$	*dom2*	t_{11}	$es_{11}\_p_1$	$es_{12}\_p_1$	100.000.000	5000	205.352
s_{13}	$s_6, s_8, s_9, s_{10}, s_{11}, s_{12}$	*dom2*	t_{12}	$es_{11}\_p_1$	$es_{12}\_p_1$	100.000.000	5000	481.128
s_{14}	s_9	*dom2*	t_8	$es_8\_p_1$	$es_7\_p_1$	500.000	400	0

model is presented which is generated using the graphical modeler. Two TSN switches with each 8 ports connect 12 end-stations that are physically separated by the switches. All links have an available bandwidth of 1 Gbit/s. Two application domains are defined which contain 12 topics. Using the Prolog 3.3 rule, 14 streams are derived which carry these topics from publishers to consumers. The modeled topics are all periodic and have timing requirements and must be scheduled using the IEEE 802.1Qbv mechanism. The timing and reliability features are visually represented with a clock and paired symbols. This and other information such as the size and period of the topics can be set in adequate dialog boxes when clicking on each topic.

To avoid design mistakes, specific mechanisms are implemented to avoid logically impossible situations. For instance, when a new Ethernet link is being created via drag&drop, it is not allowed to connect it to an already connected port. The details about streams, topics, non-overlapping relations of the streams, and the synthesized schedule driver from the framework are presented in Table 4.1. The stream's gate opening values are all in a correct range between 0 and the streams' periods. The computed first gate opening time is used in combination with Algorithm 3 to compute the GCL tables. For this case study, the framework is installed on a desktop PC with an Intel Core i7 CPU and 4GB RAM. For the synthesis, the SMT solver Z3 Version 4.5.0 ist used.

4.7.3 Discussion

The presented framework consists of different architectural modules (Figure 4.9) in order to address the mentioned challenges. The modularity and extensibility are essential requirements

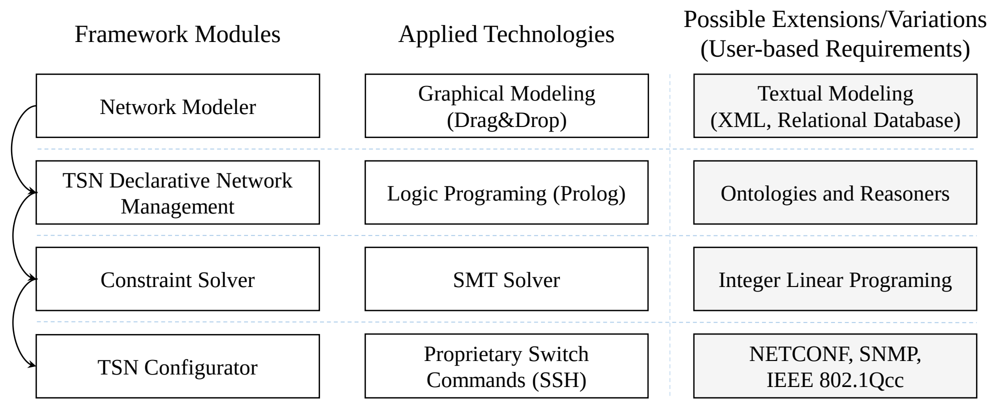

Framework Modules
Applied Technologies
Possible Extensions/Variations (User-based Requirements)
Network Modeler
Graphical Modeling (Drag&Drop)
Textual Modeling (XML, Relational Database)
TSN Declarative Network Management
Logic Programing (Prolog)
Ontologies and Reasoners
Constraint Solver
SMT Solver
Integer Linear Programing
TSN Configurator
Proprietary Switch Commands (SSH)
NETCONF, SNMP, IEEE 802.1Qcc

Figure 4.9: Modularity and extensibility of the developed framework

for future TSN developments. It has to be guaranteed that the implemented modules can be modified, enhanced or replaced separately with minimum impact on the rest of the modules. Considering the network modeling module, there can be other requirements coming from framework users. Although the graphical modeling approach can is helpful for small to medium sized networks, for very large networks textual or table-based modeling could be required. Such extensions can be easily implemented and added to the framework. It is possible because of the plug-in-based nature of EMF, which is used for implementation of the framework in this thesis.

To implement the TSN declarative network management module, Prolog is applied. Based on framework user requirements, the presented approach can be extended by the integration of ontologies and reasoners. To build an ontology, a transformation module has to be implemented to convert the graphical object-oriented model to an ontology. An ontology-based approach for modeling of TSN networks has been developed and presented in our previous work [142].

The scheduling constraints can be solved by different solver implementations such as SMT (which is used in our framework) or ILP solvers. Factors such as licensing costs, performance (synthesis speed), and additional features such as constraint labeling (a required feature in this thesis) are significant to select an adequate solver. The SMT solver used in this thesis can be easily replaced by other solvers. To achieve this, only the constraint generation algorithm and its implementation (cf. Algorithm 2) has to be adapted.

After the scheduling synthesis is done, the results should be used to configure the GCL tables (cf. Algorithm 3). Such a configuration has to be distributed among all relevant TSN switches in the network where the schedule of egress ports have to updated. There are different options to execute this task. An initial approach is to use the proprietary configuration commands which can be executed via Secure Shell (SSH). Alternatives are NETCONF protocol [153] and Simple Network Management Protocol (SNMP) [154] which are being integrated into the TSN standard IEEE 802.1Qcc to simplify and automate the configuration procedure. As soon as the switch manufacturers provide these option for TSN switch ports, the schedule synthesis results can be distributed by these protocols.

Chapter 5

Model Correction and Optimization

In this chapter, the focus is on situations where a designed network model is not satisfiable and no feasible schedules are found by the solver. In contrast to simple network models, localization and correcting unsatisfiability of complex network models is time-consuming and increases the development costs. To tackle this problem, a constraint labeling approach is presented which aims to assist the designers to better understand where in the model the unsatisfiability occurs i. e. there are conflicting constraints that cannot be satisfied at the same time.

Only if the solving procedure terminates within acceptable time (depending on engineering requirements), network configuration or model correction can be started. To optimize the model before synthesis, Prolog-based queries are presented which aim at reducing the synthesis time by optimizing the network model. The presented solution exploits the power of logic programming to formulate adequate queries before schedule synthesis where the responses can be used to optimize the model and avoid e. g. bottlenecks.

5.1 Problem Localization in Unsatisfiable Models

The goal of schedule synthesis for IEEE 802.1Qbv is to find a feasible time-triggered schedule. To solve this problem, scheduling constraints (based on the designed network model) must be satisfied and stream offsets have to be calculated. A network model consisting of constraints is unsatisfiable if it contains at least two conflicting constraints. We call a network model infeasible if the conflicting constraints are generated because of scheduling requirements. To find network and application segments causing the infeasibility in the model, we exploit the concept of unsatisfiable cores which can be produced by SMT solvers.

5.1.1 Background: Unsatisfiable Cores

A formula in Conjunctive Normal Form (CNF) is defined a set of clauses, where each clause itself contains literals [155]. A literal is a variable or the negation of it. A CNF formula is satisfied under an variable assignment, if each clause is satisfied. Accordingly, a clause is satisfied if at least one of its literals can be satisfied. The task of a SAT solver is to prove the unsatisfiability of a CNF formula or to compute a satisfying variable assignment (e. g. finding time-triggered stream offsets of a feasible schedule). If a CNF is unsatisfiable, the formula is called contradictory. Any subset of the clauses in such a CNF formula that is still unsatisfiable is defined as an unsatisfiable core (*unsat − core*). We consider the following formula with integer interpretation in SMT as an example.

$$\underbrace{(x \geq 0)}_{C1} \wedge \underbrace{(y \geq 0)}_{C2} \wedge \underbrace{(y < 6)}_{C3} \wedge \underbrace{(x > 2)}_{C4} \wedge \underbrace{(y > x - 2)}_{C5} \wedge \underbrace{\neg(y \geq -x + 8)}_{C6} \wedge \underbrace{(x < 3)}_{C7}$$

The *unsat − core* can be extracted using the following:

```
    (set-option :produce-unsat-cores true)
    (declare-const x Int)
    (declare-const y Int)
    (assert (! (>= x 0):named C1))
    (assert (! (>= y 0):named C2))
    (assert (! (< y 6):named C3))
    (assert (! (> x 2):named C4))
    (assert (! (> y (- x 2)):named C5))
    (assert (! (not (>= y (+ (* x -1) 8))):named C6))
    (assert (! (< x 3):named C7))
    (check-sat)
    (get-unsat-core)

```

The satisfiability is checked with the following output indicating that the formula is contradictory because of conflicting C4 and C7 clauses.

```
unsat (C4 C7)
```

Many applications of constraint systems deal with finding satisfying variable assignments for satisfiable constraint sets. Hence, a big number of algorithms aiming at finding satisfying assignments [156]. For unsatisfiable constraint sets, tools and algorithms are required for infeasibility analysis. According to [156], there are two interesting questions which have to be answered by such algorithms: (i) How many constraints can be satisfied? and (ii) Where is the origin of the unsatisfiability in the set of constraints?

The second question is the interesting one in the context of this thesis. Solutions presented for this question intend to find Minimally Unsatisfiable Core (MUC) or Minimally Unsatisfiable Subsets (MUSes) for a given constraint set. A new algorithm for infeasibility analysis that quickly enumerates MUSes is called MARCO and implemented in [157].

Z3 produces an unsatisfiable core but does not try to minimize it. A method and tool called HSMTMUC are presented by [158] aims at finding MUC for Z3. The presented approach is based on the deletion-based MUC extraction method [159]. It is implemented and available on [160]. The performance of the approach is evaluated using hundreds of benchmarks [158].

5.1.2 Approach

The presented approach consists of two parts. The first part deals with an adequate constraint labeling approach. The second part introduces an iterative correction method which exploits the labels of the conflicting constraints to localize the design errors in the model. Designers who work on an erroneous segment of the model (considering both network topology and application) must to check the labels and estimate how to modify the model towards satisfiability.

5.1.2.1 Constraint Labeling

Considering an unsatisfiable set of constraints, user-friendly feedback is required when it comes to detecting the high-level modeling decisions which cause the low-level constraint contradictions [161].

The main goal of the constraint labeling approach is to identify fragments of the network which are involved in the conflicting constraints. Answering the following questions are required:

- Why is a specific constraint generated?
- Which topological hardware components (e. g. switch ports) are involved?
- Which software components are involved?

To answer the first question, a classification of TSN constraint types is required which is presented in Figure 4.1. It helps to understand which TSN standard caused the problem. Based on the available expert knowledge about each standard and its details, the designer can get closer to the origin of the problem.

For example, all non-overlapping constraints generated based on IEEE 802.1Qbv belong to the same constraint class. Considering e. g. the non-overlapping constraint, a large stream with arbitrary period together with a smaller stream with a very short period can lead to unsatisfiability. The reason is that the transmission time of the large stream can take longer than the period of the smaller stream. In this case, the smaller stream misses its reserved time-triggered slot. Using such an expert knowledge, developer(s) who are working on time-triggered streams, can modify the model e. g. by re-routing of streams or topology changes, to find a feasible schedule. In contrast, developers working on reliability features regarding IEEE 802.1CB are not supposed to be involved in the model correction procedures at this step when no conflicting constraints are labeled by this constraint class.

Responding to the second question about the involved hardware components, it is required for model correction to find out which nodes cause the problem and whether it is a distributed problem (many nodes and links are affected). It helps also to estimate the correction effort. For instance, if nodes in different network segments and application domains are involved, many developers are supposed to collaborate for model correction. For local and domain-specific problems, the correction overhead is less. Considering the time-triggered streams as an example, two or more streams can have conflicting non-overlapping constraints. Knowing on which network links (egress ports) these streams have conflict is significant to understand where exactly a topology modification in the model is helpful.

The information about the involved software components is required to backtrace applications that are producing conflicting constraints. Using this information together with information about the involved nodes, model correction steps can be derived by the developers (e. g. migrate an application from one node to another one or modification of the period or size of the application streams).

The labeling structure of the constraints has to be defined based on the expert knowledge about each specific TSN standard. For this purpose, a workflow is proposed in Figure 5.1. The main target group of the labeling approach is developers who model the network and must modify an unsatisfiable model. To understand each generated constraint required for correcting modifications, the most crucial information has to be selected from the model and be used to create adequate labels. These labels must be reviewed and tested by experts (TSN standard experts which deal with modeling and developing). If there are missing information or deceptive

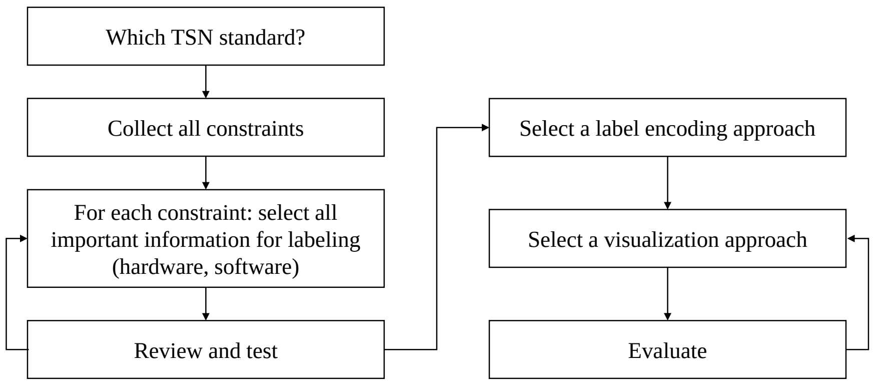

Which TSN standard?
Collect all constraints
For each constraint: select all important information for labeling (hardware, software)
Review and test
Select a label encoding approach
Select a visualization approach
Evaluate

Figure 5.1: Work flow for developers to identify and label significant information for each constraint type which help to better understanding of model unsatisfiability.

information in the labels, the information selection for labeling has to be improved iteratively. The labels must be encoded to be readable for computers. For instance, JSON or XML encoding technologies can be applied. The encoded labels are used for visualization purposes and help to analyze the constraints and backtracing the design errors of the model.

Taking IEEE 802.1Qbv and the non-overlapping constraint as an example, the information related to the time-triggered streams is required to build the labels. The labels related to this constraint are expected to contain information about the data topic and application domain which are transmitted, the period and size of the data topic, the name of the publisher and consumer end-stations, and the name of all egress ports on the path. Demonstrating examples are presented in Section 5.1.3.

5.1.2.2 Iterative Correction

Based on the constraint labeling approach and the available algorithms to compute minimal satisfiable cores, a general approach for iterative model correction is proposed in Figure 5.2. The generated network model is the input for three SMT solver algorithms which can compute (minimal) $unsat - cores$: (i) MARCO [156], (ii) HsmtMuc [158], and (iii) standard Z3 implementation (the computed $unsat - core$ is not necessarily minimal). As the computation time of these algorithms is unknown, a specific amount of time is defined as a timeout to avoid infinite run-time. If all constraints are satisfied, the correction process terminates and network configuration process will be started based on the synthesized configuration parameters such as time-triggered offsets for IEEE 802.1Qbv. However, if the mentioned algorithms find

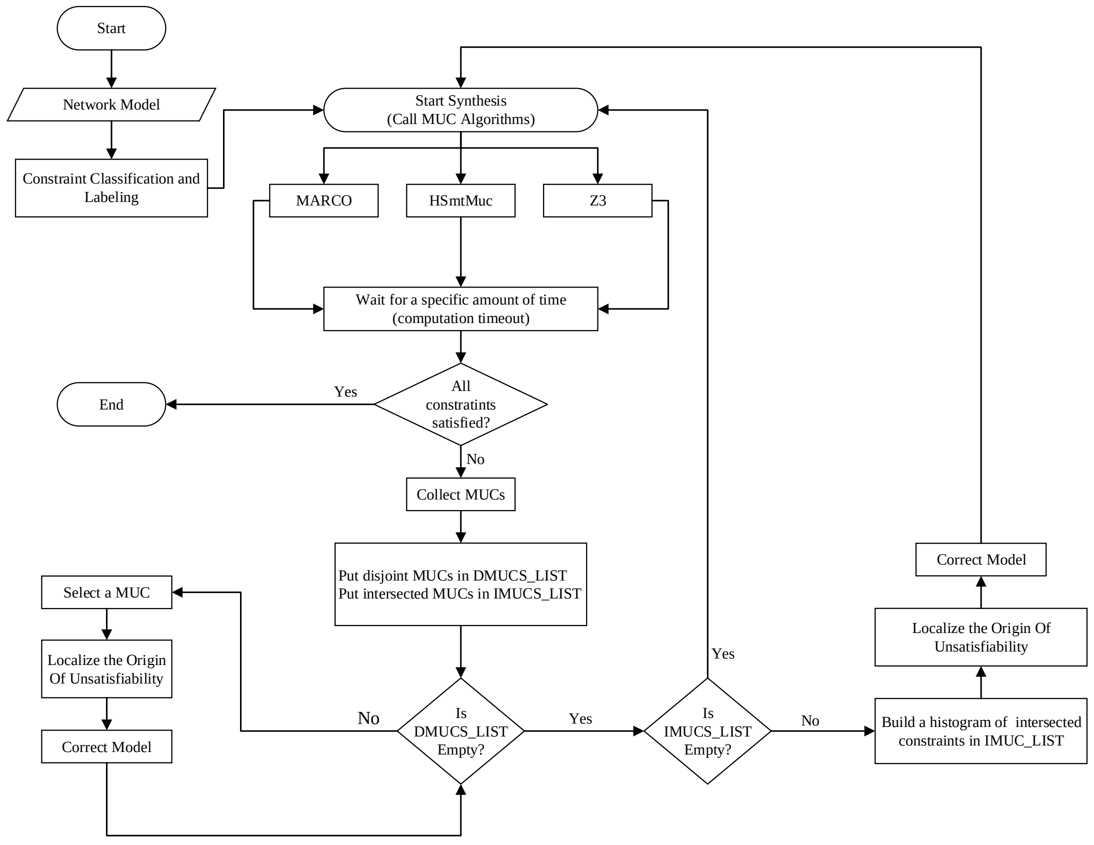

Start
Network Model
Constraint Classification and Labeling
Start Synthesis (Call MUC Algorithms)
MARCO
HSmtMuc
Z3
Wait for a specific amount of time (computation timeout)
All constraints satisfied?
Yes
End
No
Collect MUCs
Put disjoint MUCs in DMUCS_LIST
Put intersected MUCs in IMUCS_LIST
Is DMUCS_LIST Empty?
No
Yes
Select a MUC
Localize the Origin Of Unsatisfiability
Correct Model
Is IMUCS_LIST Empty?
Yes
No
Build a histogram of intersected constraints in IMUC_LIST
Localize the Origin Of Unsatisfiability
Correct Model

Figure 5.2: Overview of the model correction approach using the unsatisfiable cores.

$unsat - cores$ for the model, the conflicting constraints have to be analyzed. Different algorithms may find different $unsat - cores$ for the same model. Some of these cores are disjoint, which means they do not share any constraint with other cores, or they are intersected (cores which share at least one constraint). Disjoint cores are put into a list called DMUCS_LIST, and the intersected cores are collected in IMUCS_LIST.

We start with DMUCS_LIST because disjoint cores are rather pointing to a local problem (do not share any constraints which can mean they come from different network segments). For each core in this list, the developers try to localize the problem and derive correcting actions. It will be repeated until the list is empty.

To analyze the intersected cores in IMUCS_LIST, we propose to build a histogram of constraint labels as a tool to measure the importance of the intersected constraints. Constraints with a high number of incidence can point to a major problem in the model. It must be localized and corrected with higher priority. After correcting the model, the synthesis process is started

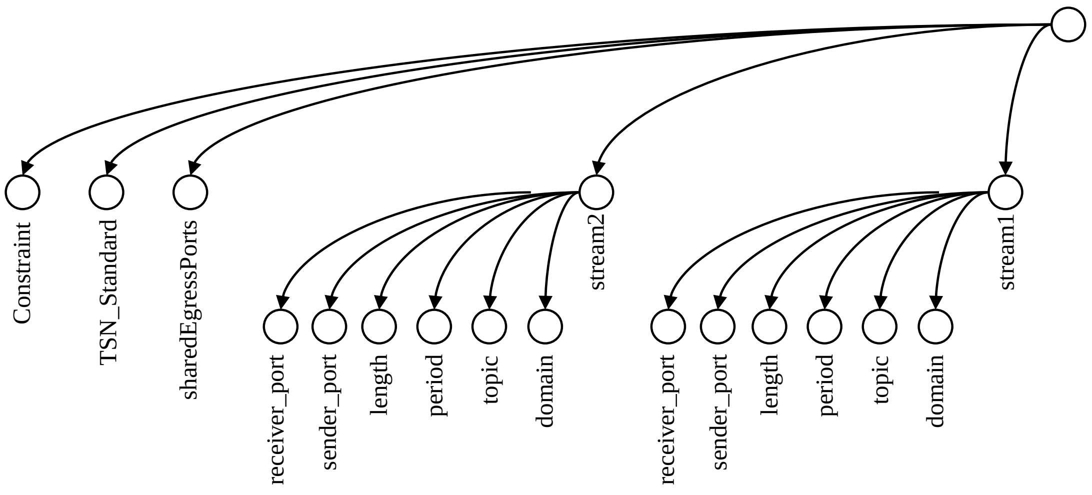


Figure 5.3: Label structure (variables) of non-overlapping constraint in TSN IEEE 802.1Qbv standard.

again. As model corrections themselves can lead to new satisfiability problems, the correction is iteratively executed until the model is satisfied.

For instance, we consider the (i) offset < period and (ii) non-overlapping constraints which are presented in Figure 4.1. The label of the first constraint contains information about the TSN standard name *TSN_Standard*, the identification of the *Constraint*, and stream information i. e. application domain, topic, period, length, and the ports name of the receiving and sending end-stations. The non-overlapping constraints always contain two streams which have conflicts (no feasible schedule found that guaranteed the non-overlapping requirement). Moreover, it contains a list of shared egress ports *sharedEgressPorts* between two conflicting streams which can help to better understand the unsatisfiability of the model. In Figure 5.3, the structure of non-overlapping labels are visualized.

The localization of the problem can be achieved with the analysis of the labels pointing to hardware and application segments in the network. After localization, the problematic model configuration must be inferred (by the developers). For simple and obvious problems where e. g. only two network streams and few nodes are affected, reviewing the labels can directly lead to a *good* problem explanation. For more complex problems with many affected segments, this process can be more complicated and extra information about the model is required. A constraint label cannot contain information about all other network components which can play a role regarding the model unsatisfiability.

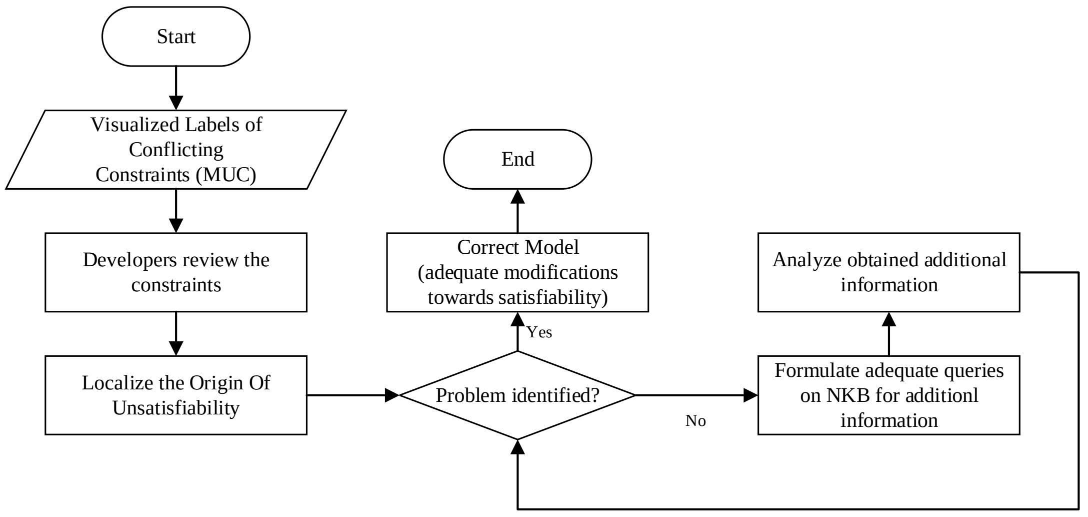

Start
Visualized Labels of Conflicting Constraints (MUC)
Developers review the constraints
Localize the Origin Of Unsatisfiability
Problem identified?
Yes
No
Correct Model (adequate modifications towards satisfiability)
End
Formulate adequate queries on NKB for additionl information
Analyze obtained additional information

Figure 5.4: Overview of the model correction approach using the unsatisfiable cores.

Hence, the developed Prolog-based NKB can be used (Figure 5.4). If the developers localize the problem for instance in a specific egress port of a switch, adequate Prolog queries can be formulated to obtain additional information about other network components close or relevant to this egress port. For each specific TSN standard, a different type of additional information is required.

5.1.3 Evaluation

To evaluate the usage of the labeled constraints for model correction, a complex network model is developed which contains four main application domains (Figure 5.5). Each domain has a set of periodic topics with real-time requirements which are transmitted within separated network segments. The objective is to compute a feasible schedule for all relevant egress ports in the network. The details about the topics are presented in Prolog A.3. For the sake of convenience, only the native Z3 feature is used to produce the unsatisfiable core. ($unsat - core$). Following the presented approach in Figure 5.1, JSON is used to encode and decode the labels.

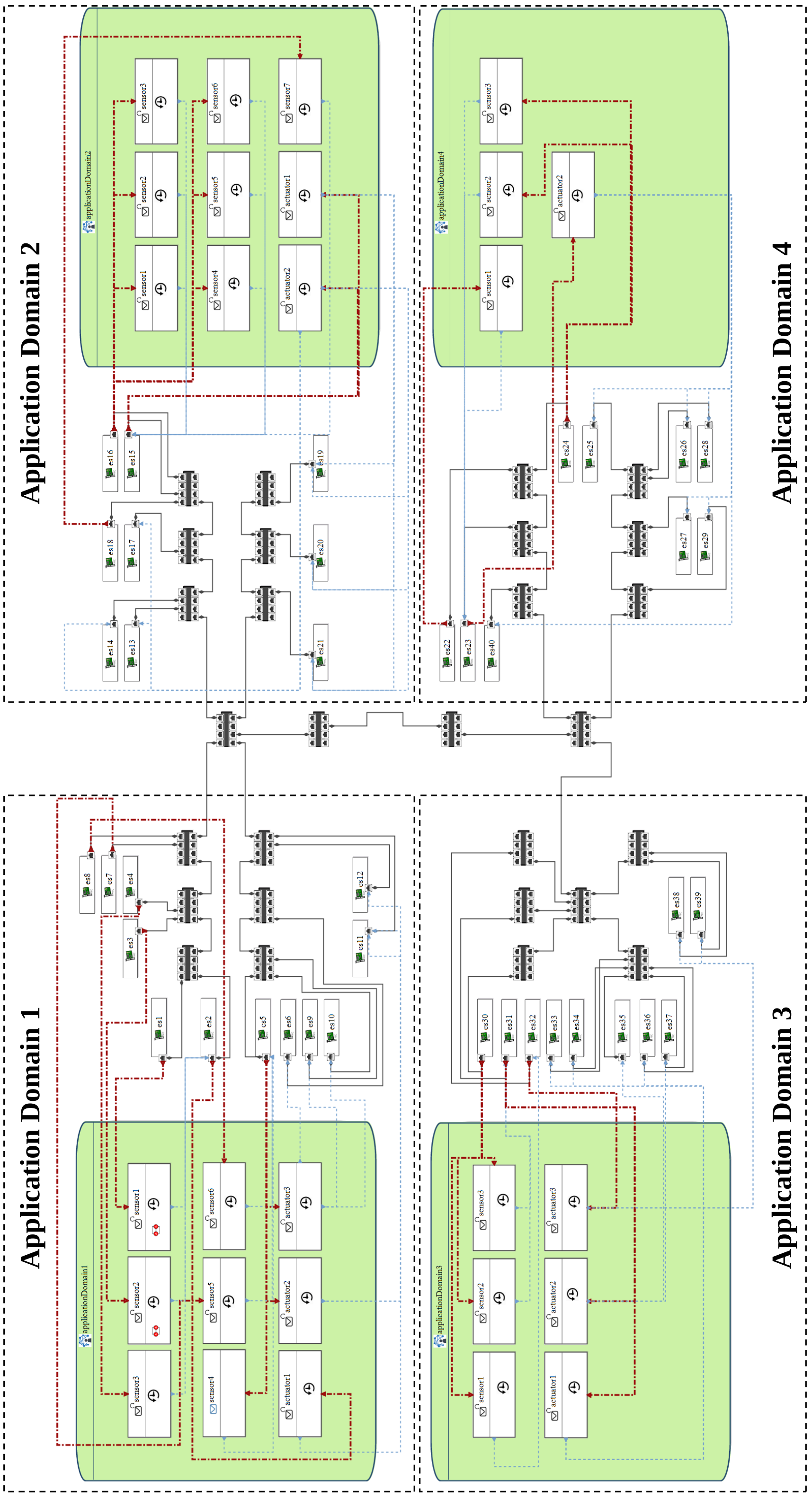


Figure 5.5: A complex unsatisfiable network model is designed. The unsatisfiability reasons are located in application domains 1 and 4. The model can be corrected using the labeled constraints. For more details see Prolog A.3, Figure A.1, and Figure A.2.

.

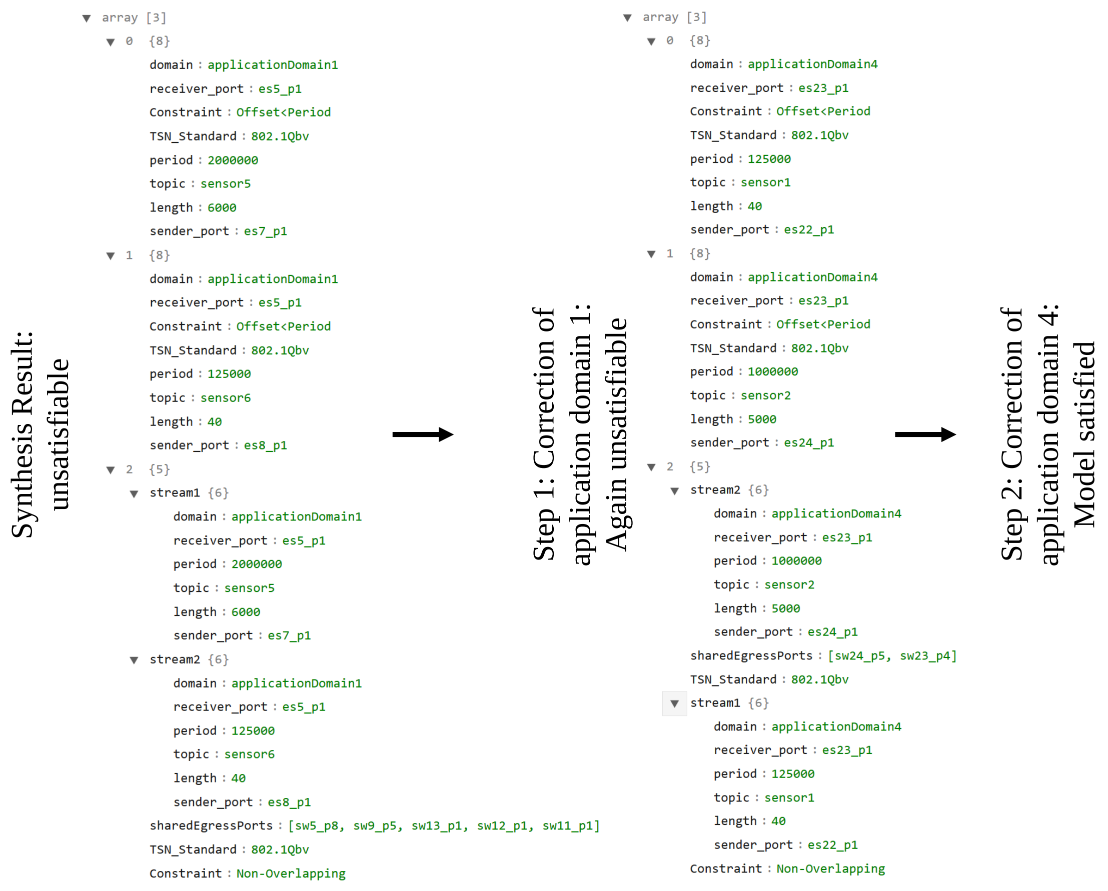
After the execution of the synthesis algorithm, the framework indicates that the model is not satisfiable and produces an unsatisfiable core pointing to a problem in the application domain 1. Two streams have conflicts. One stream has a length of 6000 Bytes, a period of 2 ms and transports the topic *sensor5* from *es7* to *es5*. The second stream, however, has a period of 125μs and a length of 40 Bytes from *es8* to *es5*. These two streams have five shared egress ports on their path. Based on expert knowledge, the transmission delay caused by the stream of bigger size is too long for the short period of the second stream. In other words, the stream with a shorter period does not have any chance to get a free slot while the other stream is transmitted. Thus, it exceeds its period which leads to unsatisfiability.

Considering the domain application 1 and its topology, different possible solutions can be suggested for model correction. The first solution is to check if the high frequency of *sensor6* is required or it can be modified (increasing the period and less frequent transmission) such that the probability of finding a free slot is increased (the solution space for the synthesis algorithm is bigger). The second solution is a to check if the stream with bigger length can be divided into smaller streams with adequate gaps to allow the more frequent stream to transmit its small data in these gaps. A third solution is a topological solution where two modifications are required. First, *es5* obtains an additional Ethernet port and second, *es7* is moved and connected to the switch where *es5* is also connected to. Following this approach, both streams do not require to compete at any egress ports. The last solution, however, is bounded to higher costs regarding the additional hardware. Hence, first, it should be tested if only the movement of *es7* can solve the problem.

As presented in Figure 5.2, the correction process is iterative. After correcting the problem in application domain 1, the synthesis is started again and the result shows that the model is still unsatisfiable. A similar problem can be identified and corrected in application domain 4. The resulting unsatisfiable cores encoded in JSON are depicted in Figure 5.6. For more complicated unsatisfiable models with a high number of conflicting constraints, graphical visualization of the cores is significantly helpful to reduce the cognitive analysis effort in case of large JSON files. However, the graphical visualization of the cores is out of the scope of this thesis.

5.2 Shortening Synthesis Time

Because the presented synthesis problem is NP-complete, it is necessary to mention the synthesis time challenges for network developments [162]. The network development can be further


Synthesis Result: unsatisfiable
Step 1: Correction of application domain 1: Again unsatisfiable
Step 2: Correction of application domain 4: Model satisfied

```
▼ array [3]
  ▼ 0  {8}
      domain : applicationDomain1
      receiver_port : es5_p1
      Constraint : Offset<Period
      TSN_Standard : 802.1Qbv
      period : 2000000
      topic : sensor5
      length : 6000
      sender_port : es7_p1
  ▼ 1  {8}
      domain : applicationDomain1
      receiver_port : es5_p1
      Constraint : Offset<Period
      TSN_Standard : 802.1Qbv
      period : 125000
      topic : sensor6
      length : 40
      sender_port : es8_p1
  ▼ 2  {5}
    ▼ stream1 {6}
        domain : applicationDomain1
        receiver_port : es5_p1
        period : 2000000
        topic : sensor5
        length : 6000
        sender_port : es7_p1
    ▼ stream2 {6}
        domain : applicationDomain1
        receiver_port : es5_p1
        period : 125000
        topic : sensor6
        length : 40
        sender_port : es8_p1
      sharedEgressPorts : [sw5_p8, sw9_p5, sw13_p1, sw12_p1, sw11_p1]
      TSN_Standard : 802.1Qbv
      Constraint : Non-Overlapping
```

```
▼ array [3]
  ▼ 0  {8}
      domain : applicationDomain4
      receiver_port : es23_p1
      Constraint : Offset<Period
      TSN_Standard : 802.1Qbv
      period : 125000
      topic : sensor1
      length : 40
      sender_port : es22_p1
  ▼ 1  {8}
      domain : applicationDomain4
      receiver_port : es23_p1
      Constraint : Offset<Period
      TSN_Standard : 802.1Qbv
      period : 1000000
      topic : sensor2
      length : 5000
      sender_port : es24_p1
  ▼ 2  {5}
    ▼ stream2 {6}
        domain : applicationDomain4
        receiver_port : es23_p1
        period : 1000000
        topic : sensor2
        length : 5000
        sender_port : es24_p1
      sharedEgressPorts : [sw24_p5, sw23_p4]
      TSN_Standard : 802.1Qbv
    ▼ stream1 {6}
        domain : applicationDomain4
        receiver_port : es23_p1
        period : 125000
        topic : sensor1
        length : 40
        sender_port : es22_p1
      Constraint : Non-Overlapping
```

Figure 5.6: Step-by-Step analysis of unsatisfiable cores produced by Z3 solver and correcting the model towards satisfiability. The cores indicates that the problem lies on application domains 1 and 4 where time-triggered streams cannot be scheduled.

progress, if and only if the designers get an answer from the framework regarding the synthesis results. Depending on the results (satisfied/unsatisfiable) the designers can either configure the network nodes or try to understand the design errors. However, if the synthesis takes undefined long time intervals, the development cost increases for dynamic network architectures where nodes join and leave the network frequently. Thus, we first discuss the decelerating factors and also propose how the NKB (Section 3.3) can be used to optimize the network model regarding the synthesis time.

5.2.1 Decelerating Factors

Considering Algorithm 2, the most expensive operations are located in Line 21 where non-overlapping slots must be found for each stream. The number of non-overlapping streams increases when the overall number of streams increases in the model. Although it does not necessarily lead to more non-overlapping streams in all network segments, however, the probability of more competition between streams will be increased depending on the concrete configuration. Each new stream at any egress port will increase the synthesis time, because it has to be considered for a correct schedule.

The difference of streams' periods (i. e. streams with long period compete with streams with a short period) are directly relevant for computation of hyper period at any egress port. It leads to the expansion of search space for a feasible schedule (Line 24). It must be noticed that increasing the search space does not mean also a higher probability of finding a feasible schedule. Similarly, long hyper periods at egress ports increase the number of iterations (cf. [30]).

Another factor which shrinks the solution space and extends the synthesis time is a situation wherein a set of competing streams, the size and period of these streams have major deviations similar to the mentioned example of application domain 1 and 4 in Section 5.1.3. In those scenarios, the deviation is too big and leads directly to unsatisfiability because the stream with shorter period do not match in any slots (slots all are occupied because of the large size of the competing stream). However, for lower deviations, where a solution could be possible, the solution space can be extremely constricted and the algorithm requires more time to find a solution or indicates that the model is unsatisfiable.

To test the effect of the mentioned factors, the example presented in Figure 5.5 is used. We first start with the synthesis of application domain 1 and add the other domains step-by-step which increase the number of streams and also competitions on egress ports in the overall network architecture. The result of this initial benchmark is depicted in Table 5.1 where the relation between the number of streams and the synthesis time is presented.

Table 5.1: The synthesis time is increased with the increasing complexity of the competing streams based on the network model in Figure 5.5.

Application domain	Number of competing stream pairs	Number of generated scheduling constraints	Synthesis time (s)
1	17	97	0.2039
2	57	792	1.625
3	14	168	0.3359
4	18	235	0.4900
1,2,3,4	106	1292	2.7039

```
%Rule to find the list of all overloaded ports
verifyOverloading(AllowedNumber,AllOverloadedPorts):-
findall(OverloadedPort,(listAllPorts(AllPorts),
checkAllPorts(AllPorts,AllowedNumber,OverloadedPort)),
AllOverloadedPorts).
%Rule to go recursively through all network ports
checkAllPorts([H|T],AllowedNumber,Ports):-
isPortOverloaded(H,AllowedNumber,Ports);
checkAllPorts(T,AllowedNumber,Ports).
%Rule to check if a given port is overloaded
isPortOverloaded(Port,AllowedNumber,Ports):-
findall(Name,(streamFinder(Name,_,_,_,EgressPorts,_,_,_),
member(Port,EgressPorts)),EgressStreams),
length(EgressStreams,Nr), Nr > AllowedNumber,Ports=Port.
```

Prolog 5.1: Rules for verification of exisitng of overloaded egress ports. Overloading leads to longer synthesis time.

5.2.2 Model Optimization

Considering the mentioned situations decelerating the synthesis, the NKB can be exploited to avoid them. We present two new rules which verify the model to ensure specific model qualities dealing with (i) controlling and limiting the number of competing streams at egress ports and (ii) avoiding major deviations in streams' periods and size at any egress ports. The presented logic rule in Prolog 5.1 targets at finding the overloaded ports with a high number of outgoing (egress) streams. For each specific port in the network model, all streams are extracted which contain that port in their paths. The number of maximum allowed outgoing streams is defined as *AllowedNumber*. The result of traversing the model with *verifyOverloading* rule is a list of all ports which have a number of outgoing streams higher than *AllowedNumber*. The rule *checkAllPorts* recursively traverses all nodes using a logical or ($; \equiv \vee$) to avoid early stopping the search. Developers exploit this information and achieve a better load balancing to reduce the overall density of the competing streams at ports. Fewer competitions mean fewer constraints which have to be solved and leads to shorter synthesis time.

A next optimization objective before starting the synthesis is to avoid the shrinking of solution space because of significant deviations of period and size of competing streams at egress ports. For each two competing streams $S^{\mathrm{t}}_i, S^{\mathrm{t}}_j$ with periods of $S^{\mathrm{t}}_i.\rho, S^{\mathrm{t}}_j.\rho$ and transmission delay of $S^{\mathsf{t}}_{i}.\Phi_{trans}, S^{\mathsf{t}}_{j}.\Phi_{trans}$, we define the the link occupation ratio O_r as:

$$O_r = Max\left(\left(\frac{S^{\mathsf{t}}_{i}.\Phi_{trans} + S^{\mathsf{t}}_{j}.\Phi_{trans}}{S^{\mathsf{t}}_{i}.\rho}\right), \left(\frac{S^{\mathsf{t}}_{i}.\Phi_{trans} + S^{\mathsf{t}}_{j}.\Phi_{trans}}{S^{\mathsf{t}}_{j}.\rho}\right)\right) \tag{5.1}$$

Note that $S^{\mathsf{t}}_{i}, S^{\mathsf{t}}_{j}$ are not feasible, if:

$$\left(S^{\mathsf{t}}_{i}.\Phi_{trans} + S^{\mathsf{t}}_{j}.\Phi_{trans} > S^{\mathsf{t}}_{i}.\rho\right) \vee \left(S^{\mathsf{t}}_{i}.\Phi_{trans} + S^{\mathsf{t}}_{j}.\Phi_{trans} > S^{\mathsf{t}}_{j}.\rho\right) \tag{5.2}$$

which means $0 \leq O_r \leq 1$.

To increase the solution space and higher probability for a shorter synthesis time, developers can define a threshold value for O_r which must not be exceeded. The inference rules presented in Prolog 5.2, can be used to verify that all egress ports hold the defined threshold.

```
% Helper math rule to calulate the max value
storeMax(X,Y,Max) :- Max is max(X,Y) .
%Rule to find all ports with a critical link occupation ratio
verifyAllLinkOccupations(Threshold,CriticalPorts):-
findall(P,(verifyLinkOccupation(Threshold,P)),AllPorts),
sort(AllPorts,CriticalPorts).
%Rule check the ratio for each two competing streams
verifyLinkOccupation(Threshold,Port):-
overlappingStreams(N1,D1,T1,P1,L1,N2,D2,T2,P2,L2,SharedEP),
member(Port,SharedEP),
storeMax(((L1+L2)*8/P1),((L1+L2)*8/P2),O_Ratio),
O_Ratio > Threshold.
```

Prolog 5.2: Rules for verification of link occupation ratio by each competing stream pair.

Chapter 6

AutoTSN: Plug&Configure Concept for TSN

Increasing the number of distributed applications, sensors, actuators and computing units e. g. in the factory automation and automotive domain, where TSN is a potential backbone technology candidate, causes high network engineering overhead. Considering e. g. in-vehicle E/E architectures and timing requirements, network reconfiguration can be necessary if a new device must be integrated into the network architecture [142]. The presented modeling tool in this thesis automates parts of the TSN network management (scheduling and configuration). To configure the network, if a device publishes periodic and time-triggered data, a network engineer has to use the synthesis tool for rescheduling of the streams based on the IEEE 802.1Qbv standard.

Considering the current state of the TSN standards, three steps are defined towards a fully automated network configuration [163]. In the first step, the configuration of the streams must be calculated manually. The calculated parameters such as stream offsets are transmitted to the switches and end-stations using a centralized network management software interface. In the next step, tools and algorithms are expected which are used to automate the configuration tasks partially (e. g. the contribution of this thesis regarding the model-based synthesis). In a final step, all of the configuration steps must be automated. It means the vision will be that developers only plug devices containing their profiles which describe the devices' properties, their applications and timing requirements. This information is automatically collected and analyzed to compute a valid configuration (Figure 6.1).

To achieve a fully-automated configuration, it is essential that the configuring components of the network understand the application logic between the connected devices based on the

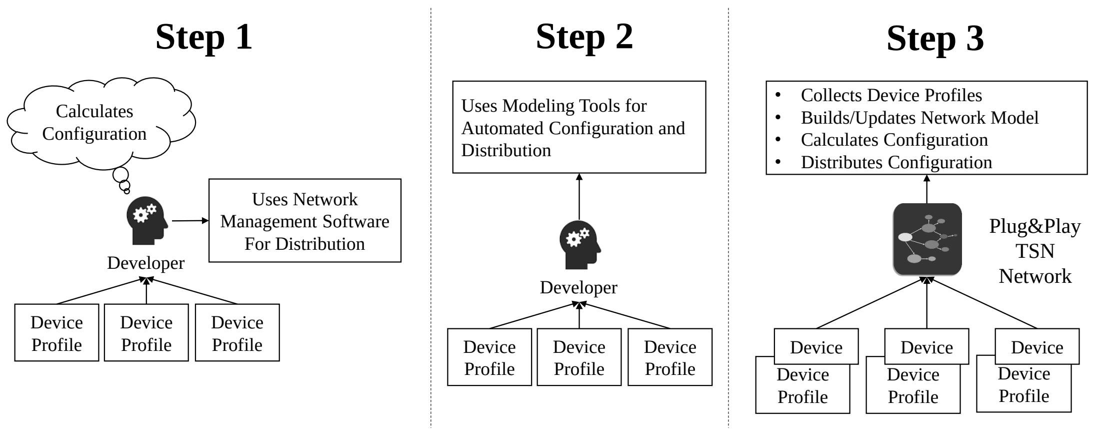


Figure 6.1: Three steps towards fully-automated configuration of TSN networks.

collected profiles. The challenge here is the fact that each device contains only a part of the application logic. For example, a new plugged monitoring device knows only that it requires/receives data from temperature or pressure sensors to monitor the system to avoid exceeding defined safe ranges. However, it does not know where the sensors are located physically in the network. Similarly, a sensor only knows that it has to send temperature values periodically but without knowing which concrete device or application intends to receive it. Putting all this information together, logical streams can be derived and used for an adequate configuration.

In this chapter, a general approach based on the publisher-subscriber paradigm is proposed to deal with the autoconfiguration of the TSN networks at run-time. It is shown, how the network knowledge base can be exploited to make configuration decisions.

6.1 Background: Publisher-Subscriber Paradigm

Mission-critical distributed applications have been traditionally implemented using a message-centric approach to transmit data between nodes of the network. This approach leads to significant inefficiencies, increases application development costs, and causes high maintenance overhead [118]. In contrast, a data-centric approach e. g. the OMG's Data Distribution Service specification follows an elementary different approach, which is explained in the following.

A major difference between message-centric and data-centric paradigm is about the state of the changes in the network. Following the message-centric approach, maintaining the state of the data objects is part of the application and the developers have to deal with it. In contrast

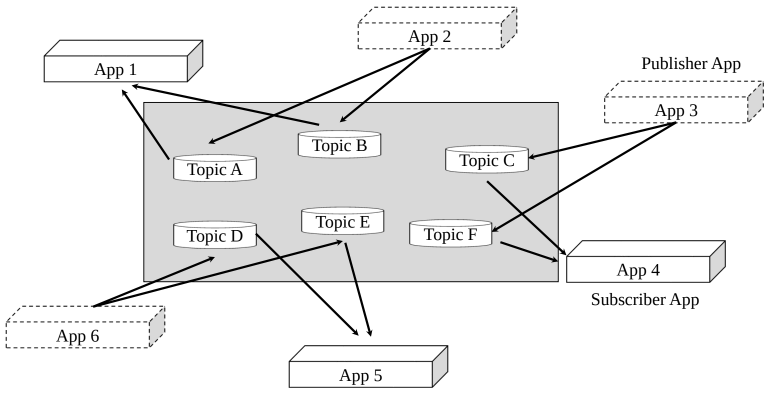


Figure 6.2: Location transparency provided by publishing and subscribing data topics by distributed applications.

and based on the data-centric concept, the networking infrastructure is responsible for data state consolidation and maintenance of changes. This complexity is hidden from the application and also the application developers. For example, considering a temperature regulation application, a developer who implements the control algorithm is only interested in the current value of the sensor temperature. Using a message-centric approach, the developer must create and send request messages to the sensor periodically and de-serialize the received responses. However, following the data-centric approach, the developer only subscribes for the current temperature value and uses it for the control application.

To provide location transparency, the publisher-subscriber concept (Figure 6.2) defines data topics which can be published or subscribed in the network. A middleware resolves the valid routes from the publisher to the subscriber(s) and takes responsibility for transmission of the data topics.

6.2 Approach Overview

The main concept of the proposed approach is to use a data-centric middleware at the application layer for dynamic networking and to combine it with the developed model-based synthesis in this thesis for configuration of the underlying TSN network.

To achieve an automated reconfiguration in TSN networks, two services are defined. A *Topology Discovery Service* is required to

- update the list of the Ethernet links after a new device joins,

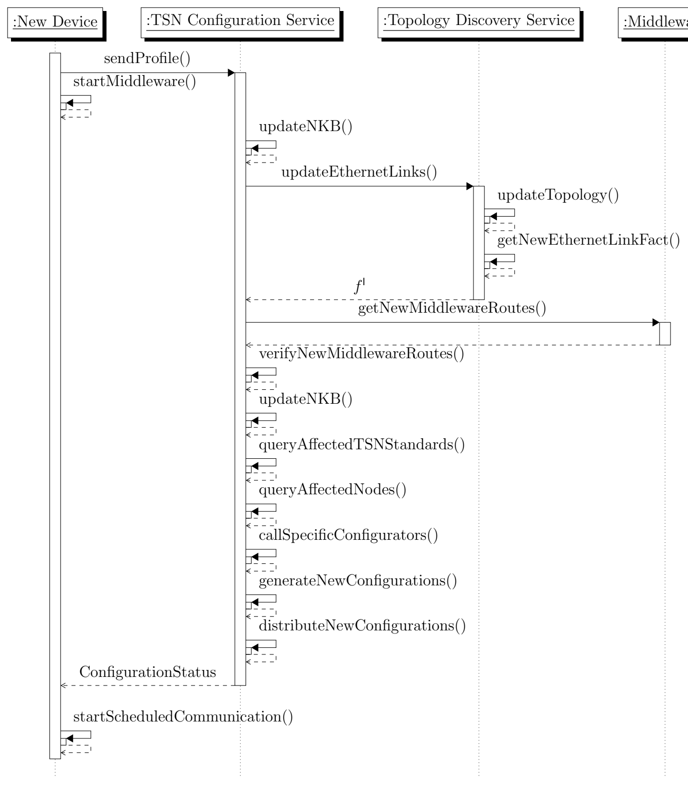

:New Device
:TSN Configuration Service
:Topology Discovery Service
:Middleware
sendProfile()
startMiddleware()
updateNKB()
updateEthernetLinks()
updateTopology()
getNewEthernetLinkFact()
f^l
getNewMiddlewareRoutes()
verifyNewMiddlewareRoutes()
updateNKB()
queryAffectedTSNStandards()
queryAffectedNodes()
callSpecificConfigurators()
generateNewConfigurations()
distributeNewConfigurations()
ConfigurationStatus
startScheduledCommunication()

Figure 6.3: A general approach for automated reconfiguration of TSN networks.

- verify the calculated routes by the middleware, and
- retrieve all Ethernet link facts f^l as defined in Section 3.3.1.

A *TSN Configuration Service* receives the device profile consisting of its facts as defined in Section 3.3.1 describing the device and its intentions (such as publishing or subscribing topics). It calls the topology discovery service to get an update of all link facts. In the next step, the service tries to figure out which TSN standards require reconfiguration using queries on the NKB. These queries are very specific for each standard.

For example, to check whether the IEEE 802.1Qbv schedule must be reconfigured for some egress ports, the configuration service creates a query to check if a newly joined device caused new periodic and time-triggered streams. If the device publishes a periodic topic with timing requirements and there is a device in the network which subscribes for this topic, then the configuration service must find the affected egress ports and call the SMT solver for a schedule synthesis. Moreover, this service has to communicate with the middleware to verify the newly calculated routes in the network. It means, using the NKB, it has to be checked if the derived streams from the NKB go through the same nodes compared to the calculated routes by the middleware for the same streams. This can be the case if there are cycles in the topology and middleware selects a route which differs route derived from the TSN model. Such a conflict leads to incorrect timing of the streams.

The new configurations are distributed to the related network nodes including end-stations and switches. If the configuration process is successful, the device starts the middleware instance which manages the application. From this point, the middleware takes control. The sequence diagram in Figure 6.3 presents the general reconfiguration steps.

6.3 Autoconfiguration of IEEE 802.1Qbv

In this section, a concrete approach for automatic reconfiguration of the TSN time-triggered mechanism is proposed and demonstrated using an example.

While a data-centric middleware is responsible for the dynamic data routing at the application layer based on a publisher-subscriber model, the developed modeling tool is used for reconfiguration of the GCL tables of th relevant TSN egress ports.

The main objective is to configure the TSN layer before starting the data transmission by the middleware. This approach provides a modular network configuration at the middleware and TSN layers. Different middleware implementations can be used based on the requirements of the network.

The assumption in this approach is: *There exists a running TSN network supporting IEEE 802.1Qbv mechanism and nodes are synchronized based on the IEEE 1588-2008 protocol. End-stations may contain multiple applications with time-triggered streams publishing or consuming topics.*

The assumption of data-centricity in dataflows is a key concept in the developed modeling tool (cf. Figure 3.2). Following Figure 6.3 , each device must announce its profile before it starts executing the middleware instance and the applications which send or receive time-triggered streams. A device profile is defined as a tuple $DP = (f^{\mathsf{e}}, F^{\mathsf{t}}, F^{\mathsf{qt}}, F^{\mathsf{qr}}, F^{\mathsf{p}}, F^{\mathsf{c}})$ where F^{t} contains the set of new topics, $F^{\mathsf{qt}}, F^{\mathsf{qr}}$ refer to sets of timing and reliability requirements, and $F^{\mathsf{c}}, F^{\mathsf{p}}$ contain all new consuming and publishing facts. As demonstrated in Figure 6.4, a new device joins the network and starts its middleware instance. The device sends its profile DP to the *IEEE 802.1Qbv Configuration Server*. This server updates the NKB adding the new facts.

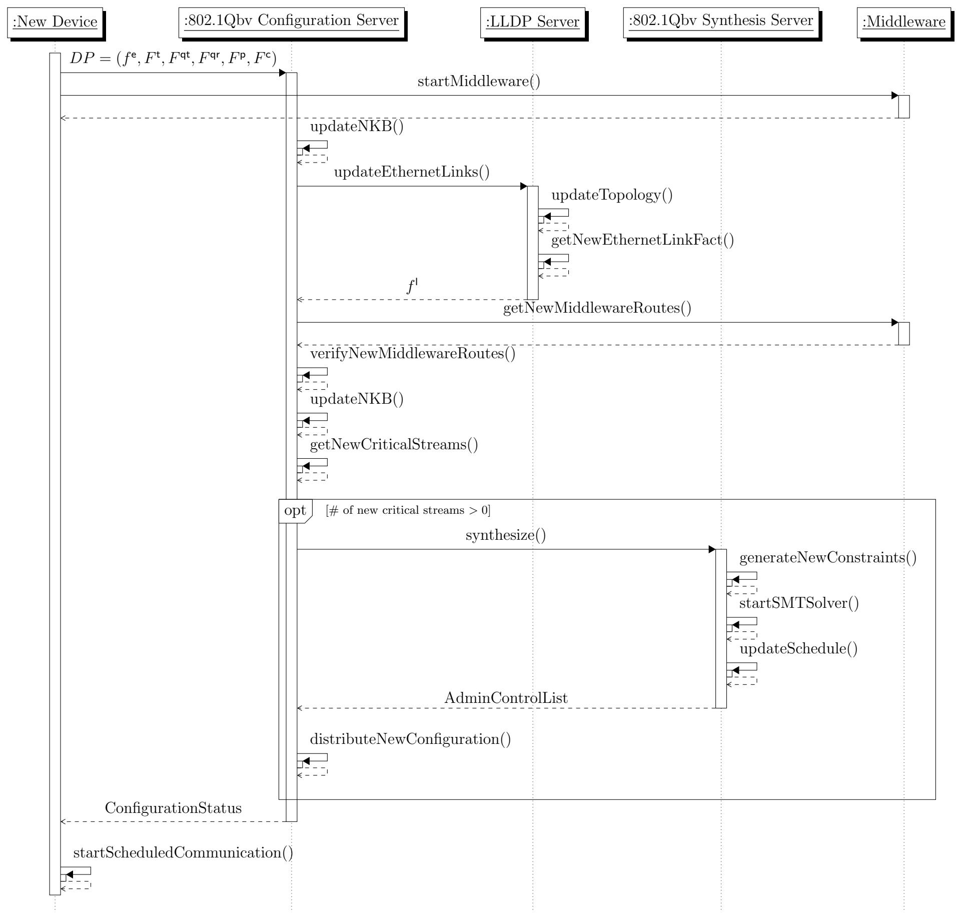


Figure 6.4: Plug&Play Procedure for the 802.1Qbv

Using e. g. *Link Layer Discovery Protocol (LLDP)* [165], the topology is discovered and the newly added Ethernet links are returned to the configuration server for updating the NKB after routes are verified.

After updating the NKB, the configuration server has to decide whether 802.1Qbv standard is affected for reconfiguration (cf. Figure 6.3) or not. To decide this, the following questions have to be answered:

- Does the new device have any periodic topics in its profile(F^{t})?
- Do they have any timing requirements (F^{qt})?
- If the new device publishes a time-triggered topic, is there any consumers for it in the network ($F^{\mathsf{p}}, F^{\mathsf{c}}$)?

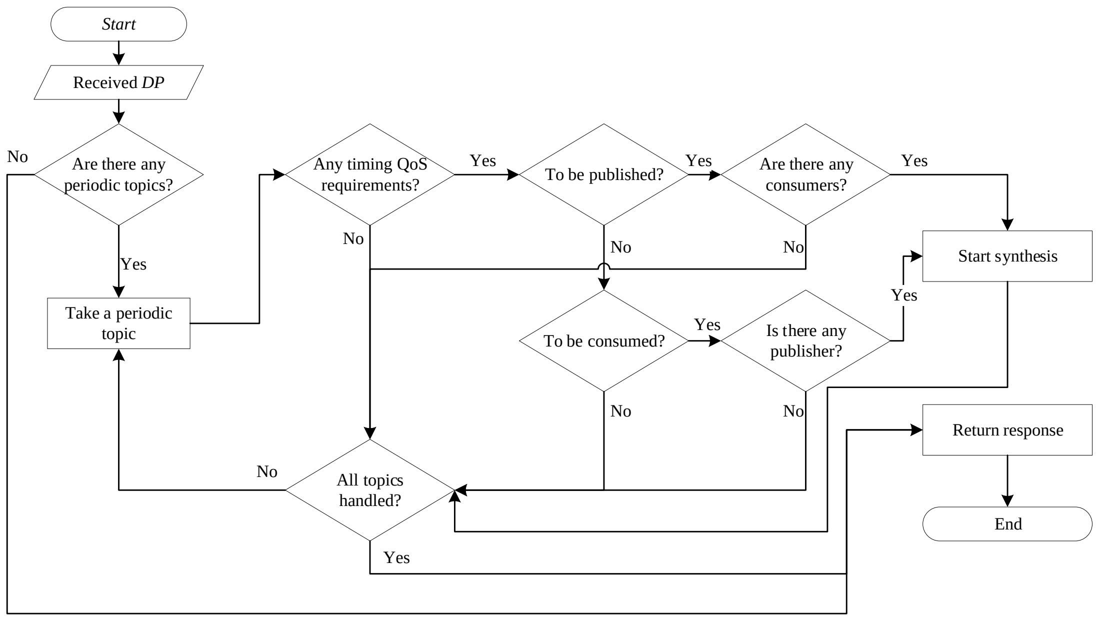


Figure 6.5: Decision procedure whether a new synthesis and reconfiguration of 802.1Qbv is required.

- If the new device consumes time-triggered topics, are they available in the network ($F^{\mathsf{p}}, F^{\mathsf{c}}$)?

Figure 6.5 depicts the flow chart explaining the decision procedure. For each new device, its profile is analyzed to find out if there are any periodic topics. For each periodic topic with timing requirement, it has to be clarified whether it must be published or be consumed. For publisher topics, there have to be active consumers in the network to have a reason for new synthesis and reconfiguration. Similarly, there has to be an active publisher of the consumed topics in the network.

If new time-triggered streams are available which compete with any other existing stream, new scheduling constraints are generated by the synthesis server and solved by the SMT solver. The synthesis server updates the schedule and sends the updated *AdminControlList* to the configuration server. It distributes the new configuration among the affected egress ports in the network.

For publisher end-stations, the *ConfigurationStatus* consists of the hyper period of the competing streams *hyperPeriod*, the synthesized offset and delay ($S^{\mathsf{t}}.tas, S^{\mathsf{t}}.\Phi_{all}$) of the stream, and its period $S^{\mathsf{t}}.\rho$. Using these parameters, the end-station calculates the sending intervals:

$$Start_k = kS^{\mathsf{t}}.\rho + S^{\mathsf{t}}.tas,$$
$$k \in [0, \ldots, (\frac{hyperPeriod}{S^{\mathsf{t}}.\rho}) - 1]$$

If there are no new time-triggered periodic streams, the NKB is updated using the new device profile and the configuration status is returned to the device as approval to further executes the middleware's procedures.

6.3.1 Adequate Prolog Rule

To decide if a new device profile causes a new synthesis or not (Figure 6.5), the NKB is exploited to avoid unnecessary and timely expensive synthesis. Even if there are new streams because of the new device, it must be verified if they have any competition with other streams on their paths. If the new device has more than one topic to publish, a new synthesis is required because of the egress port of the device itself. The outgoing streams must not overlap.

All derived new streams from the model are stored. After updating the NKB with the profile of the new device, the rule *overlappingStreams* (as explained in Prolog 3.4) is exploited and embedded in a new (rule Prolog 6.1) to verify the existence of new competing streams. The only required input for this rule consists of the domains and topics of the new device which is available in its profile.

```
newSynthesisReuqired(Domain,Topic):-
overlappingStreams(N1,Domain,Topic,P1,L1,N2,
          D2,T2,P2,L2,SharedEP).
```

Prolog 6.1: Rule to verify if new devices cause new competing streams in the network.

This efficient and short formulation justifies the feasibility and effectiveness of logic programming in modeling and querying complex relations in the network.

6.3.2 Demonstration

In this section, we demonstrate the steps of the proposed approach using a concrete example. The example consists of three applications using time-triggered streams for communication. A *Hardware Status* application is responsible to read and publish the status of its hardware with different periods. The hardware status contains the CPU load, CPU temperature, available RAM memory, and the remaining free space of its permanent memory.

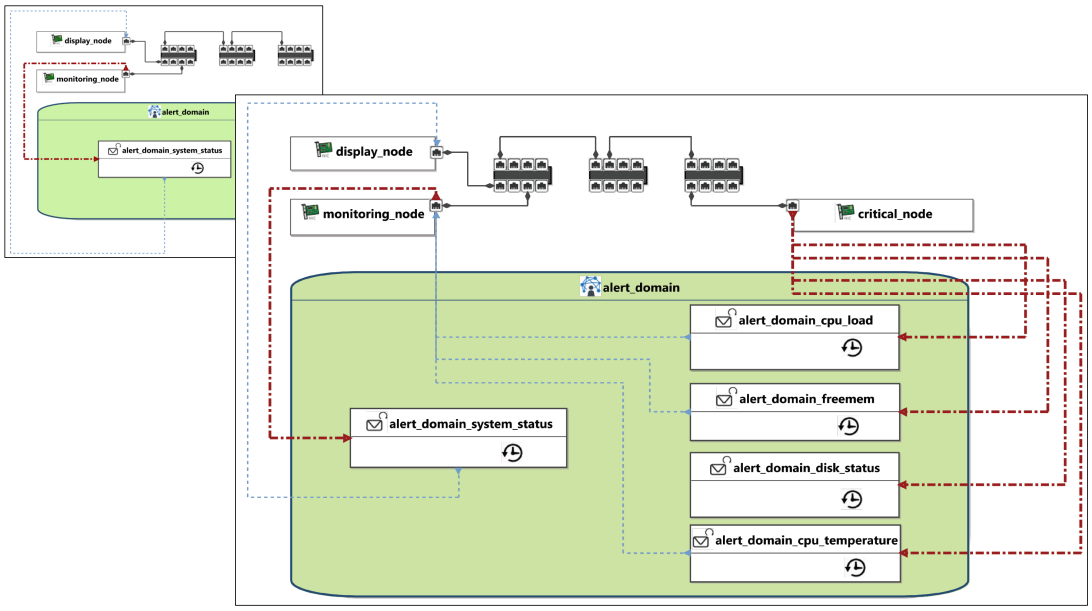


Figure 6.6: The topology of the network model before and after the critical_node is plugged in.

A *Real-time Status Monitoring* application verifies that critical nodes in the network are working appropriately and do not exceed pre-defined hardware limits such as CPU temperature. It monitors the status of the CPU load, temperature, and the RAM. Accordingly, it consumes status data of the critical nodes and publishes an overall system status (e. g. alarms) which are displayed on system monitors. The *Status Display* application consumes the system status. The applications are distributed in the network and use a data-centric middleware to manage sending and receiving the streams.

As a starting point, the monitoring and display applications are running on two embedded computing units in the network. These applications are in *alert* domain where the time-triggered *system_status* topic is published by the *monitoring* end-station and is consumed by the *display* end-station. The NKB details at this point is depicted in Prolog A.1.

After the *critical* end-station is plugged, it first registers its profile. It is physically connected to the existing end-stations over three switches (hops) supporting IEEE 802.1Qbv on their Ethernet ports. The time-triggered topics are available for potential consumers such as the monitoring application. The network model before and after plugging the *critical_node* are demonstrated in Figure 6.6. The details of the topics such as period, size, and traffic priority (with possible values from 0 to 7 as defined in IEEE 802.1p) are depicted in Table 6.1. An excerpt of corresponding new network facts are shown in Prolog A.2.

Table 6.1: Topics properties of the Plug&Play example.

Topic	Size(Byte)	Period(ns)	Priority
alert_domain_cpu_load	200	1000000	6
alert_domain_freemem	200	20000000	6
alert_domain_disk_status	200	100000000	6
alert_domain_cpu_temperature	200	1000000	6
alert_domain_system_status	800	20000000	6

To check if there is a need for reconfiguration, all new topics are checked with the presented Prolog rule. For instance, *alert_domain_disk_status* topic is checked and the NKB responds with *false.* which means that no new competing streams are derived and therefore no synthesis is required. However, verifying the topic *alert_domain_cpu_load* results in a new competing stream pair. The queries and results are presented in Prolog 6.2.

```
%Verify alert_domain_disk_status
newSynthesisReuqired(alert_domain,alert_domain_disk_status).
%Prolog tells no synthesis required
false.
%Verify alert_domain_cpu_load
newSynthesisReuqired(alert_domain,alert_domain_cpu_load).
%Prolog tells new synthesis is required
true.
%Query to get more precise information
overlappingStreams(N1,alert_domain,alert_domain_cpu_load,_,_,N2,_
    ↪ ,_,_,_,_).
% Prolog tells N1 and N2 streams are competing
N1 = "'stream':{'topic':'alert_domain_cpu_load','domain':'
    ↪ alert_domain',
'sender_port':'es2_p1','receiver_port':'es4_p1','period
    ↪ ':'1000000','length':'200'}",
N2 = "'stream':{'topic':'alert_domain_freemem','domain':'
    ↪ alert_domain',
'sender_port':'es2_p1','receiver_port':'es4_p1','period
    ↪ ':'20000000','length':'200'}".
```

Prolog 6.2: Applying the Prolog rule to decide if new topics from a new device in the network cause a new synthesis for the network configuration.

The synthesis server is called to generate the constraints and solve them Figure 6.4). Adequately, the GCL entries are updated using Algorithm 3 and the result (*AdminControlList*) is returned to the configuration server. The configuration server generates Secure Shell (SSH) [166] commands to configure the GCL tables of the affected switch ports. After the new configuration is finalized the configuration server notifies it to the new device and it starts the scheduled communication.

6.4 Implications for Future Implementation

Considering a concrete middleware implementation to realize the AutoTSN concept, the middleware must offer adequate interfaces such that the TSN configuration services obtain the required information such as the computed routes by the middleware. It is highly relevant for the egress port configuration regarding IEEE 802.1Qbv. Such interfaces have to be offered by the middleware providers.

Middleware implementations provide also their specific quality of service mechanisms. It must be investigated whether this information can be used directly to derive the required NKB facts for schedule synthesis. Moreover, the middleware must provide additional networking features to modify e. g. the VLAN tag in the Ethernet frames on demand based on the stream criticality.

Another major challenge is the timing precision of the middleware especially regarding time synchronization and precise firing of a time-triggered stream. There are internal processes of the middleware and are executed to handle specific events such as discovery service etc. The scheduling of such processes affects the timing of the processes which are responsible for sending time-triggered streams. To avoid high inaccuracy, the real-time scheduling mechanism have to be applied to fully control the behavior of all middleware processes. It is even more challenging when there are also other additional processes running on an end-station apart from the middleware processes. To provide the desired determinism, real-time operating systems can be used. In this case, it has to be investigated how such real-time operating systems and middleware implementations can be executed on hardware with limited resources considering also the boundaries of development costs.

The AutoTSN concept can be offered as a service to be integrated into the existing commercial and open-source data-centric middleware implementations.

Chapter 7

Experimental Evaluation

The objectives of this section are:

- The ability of the framework to implement a highly model-based TSN configuration is demonstrated where constraint building, schedule synthesis, and switch configuration are fully automated. We present a proof of concept that highlights how the manual work required for TSN configuration is minimized.
- The developed framework is used to model and configure an experimental setup consisting of prototypical implementation of IEEE 802.1Qbv switches. The latency and jitter of time-triggered streams are measured and analyzed regarding the expectations from IEEE 802.1Qbv.

7.1 Setup

The presented setup in this section is adapted from our previous work [167]. It is motivated by typical mixed-critical applications e. g. from automotive domain. Two synchronized step motors communicating through the network are representative for hard real-time applications with periodic behavior. Such applications require guaranteed lowest latency and jitter values. A drowsiness detection application uses the video stream of a camera to analyze the drivers' eyes and triggers alarm if it estimates that the driver has fallen asleep. An infotainment video streaming application represents that kind of applications which generate high traffic load and occupy the available bandwidth but do not have hard timing requirements. In the following, implemented applications are introduced. The details of the setup are shown in Figure 7.1.

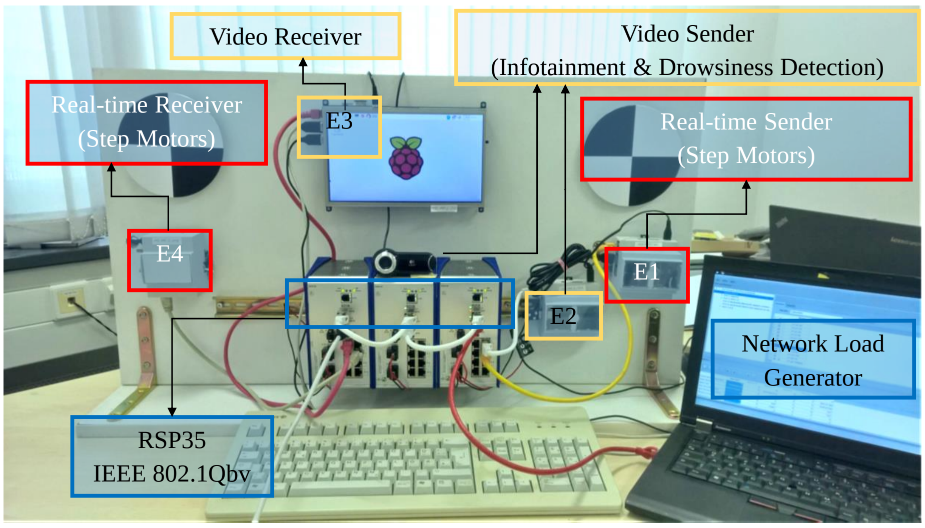


Figure 7.1: Three industrial RSP35 switches with each two prototypical FPGA-based IEEE 802.1Qbv ports are used. Automotive exemplary applications (drowsiness detection, step motor control, and infotainment video streams) are implemented on end-stations based on RaspberryPi 3 and Odroid C1 embedded hardware. The notebook generates the desired unscheduled network load to challenge the time-triggered streams.

7.1.1 Step Motor Control Application

The objective of this application is to demonstrate the importance of deterministic, low-latency, and low-jitter communication for synchronized motion control. Two Odroid-C1 (CPU: Quad-Core Amlogic S805 Cortex A5 1.5 GHz, 1GB RAM) mini-computers (with Ubuntu 14.04) are used as end-stations (E1 and E4) to control the step motors. Between these end-stations, there are three RSP35 industrial switches with prototypical FPGA-based Ethernet ports (1Gbit/s) supporting the time-triggered features of IEEE 802.1Qbv. They also support the IEEE 1588 v2 for synchronization in the nanosecond range.

After each rotation step of the right motor (E1), it sends a stream to the left motor (E4). It is time-stamped in the ingress port of the first switch (most right) and the egress port of the last switch (most left). For an initial test, the rotation and the frames are set to send data periodically each 10 ms. The frames are time-stamped by hardware in the switches. These timestamps are used to measure the latency of the received stream at E4. The clocks are synchronized using software-based (IEEE 1588 v2) at end-stations. To demonstrate the impact of correct timing in cyber-physical systems, the application is configured as follows: If the latency value is less than 6.3 μs, it is accepted by the left motor and it also rotates for one step.

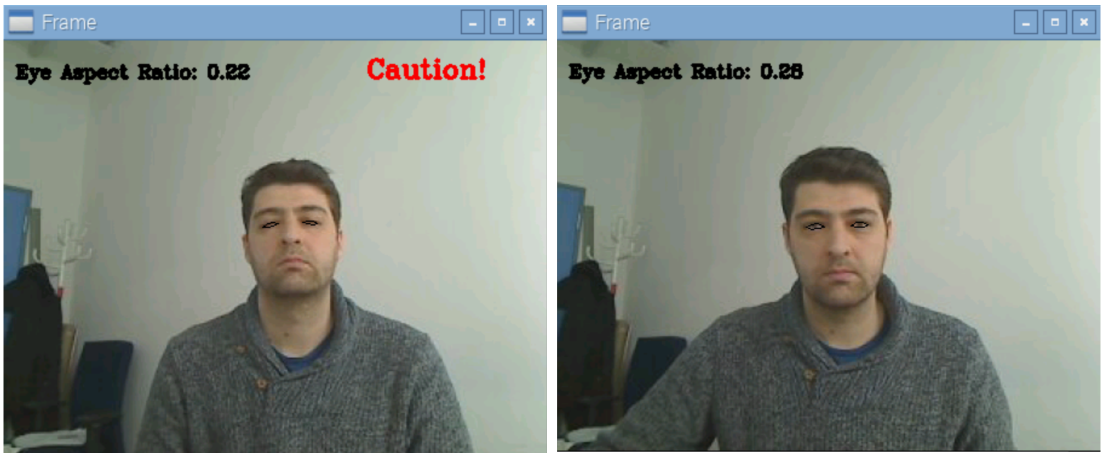
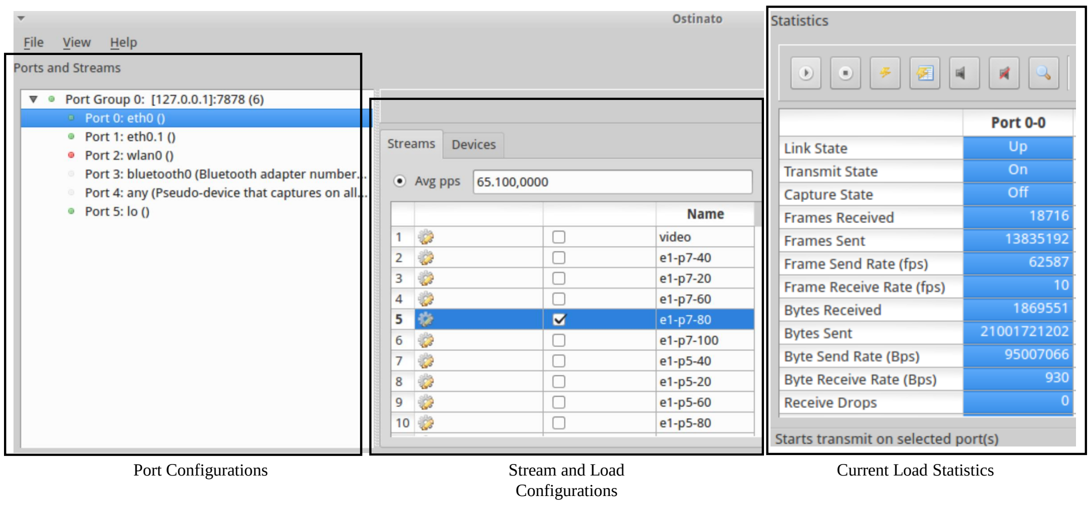


Figure 7.2: Drowsiness detector based on eye aspect ration.

This is a minimum latency we measured between the first switch ingress port and last switch egress port when no other traffic exists. Using this tight tolerable latency, the disturbance of motor movements can be demonstrated visually and acoustically if other traffic is sent through the network. As soon as other traffic (from other applications) are sent through the network and compete with the step motor streams, the step motor frames experience some delay (above 6.3 μs) which lead to misbehavior of the motors. However, when IEEE 802.1Qbv time slot reservation and guard-band mechanisms are applied, the motors rotate correctly (synchronized without any disturbance).

7.1.2 Drowsiness Detection Application

Using the video frames of a video camera which is connected to E2, a drowsiness detection application is implemented on E3. It simulates a classical video streaming application in the automotive domain. We used a prototypical implementation which is presented in [168]. It is adapted in our setup to increase the speed of detection procedure on E3. The applications run on a Raspberry Pi mini-computer (Pi 3 Model B) with a ARM Cortex-A53 CPU (1200 MHz) and 1GB RAM. The video streams occupy 2 Mbit/s of the available bandwidth of 100 Mbit/s between E2 and the first switch. The video streams and a considerable amount of additional network load are used to challenge the step motor application. Two screenshots of E3 demonstrate drowsiness application in Figure 7.2.



Ports and Streams

- Port Group 0: [127.0.0.1]:7878 (6)
  - Port 0: eth0 ()
  - Port 1: eth0.1 ()
  - Port 2: wlan0 ()
  - Port 3: bluetooth0 (Bluetooth adapter number...
  - Port 4: any (Pseudo-device that captures on all...
  - Port 5: lo ()

Streams | Devices

Avg pps 65.100,0000

			Name
1		☐	video
2		☐	e1-p7-40
3		☐	e1-p7-20
4		☐	e1-p7-60
5		✓	e1-p7-80
6		☐	e1-p7-100
7		☐	e1-p5-40
8		☐	e1-p5-20
9		☐	e1-p5-60
10		☐	e1-p5-80

Statistics

	Port 0-0
Link State	Up
Transmit State	On
Capture State	Off
Frames Received	18716
Frames Sent	13835192
Frame Send Rate (fps)	62587
Frame Receive Rate (fps)	10
Bytes Received	1869551
Bytes Sent	21001721202
Byte Send Rate (Bps)	95007066
Byte Receive Rate (Bps)	930
Receive Drops	0

Starts transmit on selected port(s)

Figure 7.3: Network load generator Ostinato [169]. Ethernet frames including flags can be configured based on experiment requirements.

7.1.3 Infotainment Video Streaming Application

Infotainment video applications can generate considerable network traffic load especially when videos with high resolution are transmitted between end-stations. Such streams have usually moderate timing requirements but require high bandwidth. The frames can be buffered without any critical consequences. If IEEE 802.1Qbv is applied, it has to be guaranteed that there is enough bandwidth for transmission of such video traffic to provide a minimum quality of service guarantee (e.g. the video on the displays should be very fluent while some artifacts can be tolerated). Such streams can destabilize the timing behavior of the critical frames when no traffic control mechanism is available. It decreases the determinism of latency and jitter of time-critical streams.

The developed application can start an arbitrary number of video streams with different resolutions from E2 to E3. The maximum achieved transmission rate is, however, bounded by the link capacity of 100 Mbit/s.

7.1.4 Network Load Generator

The presented drowsiness and infotainment applications together can generate theoretical maximum traffic of 100 Mbit/s which is equal to link capacity of E2 and E3. However, the TSN switch ports provide a bandwidth of 1 Gbit/s. To achieve a higher load challenge at TSN switch ports, a traffic generator software called Ostinato [169] is used.

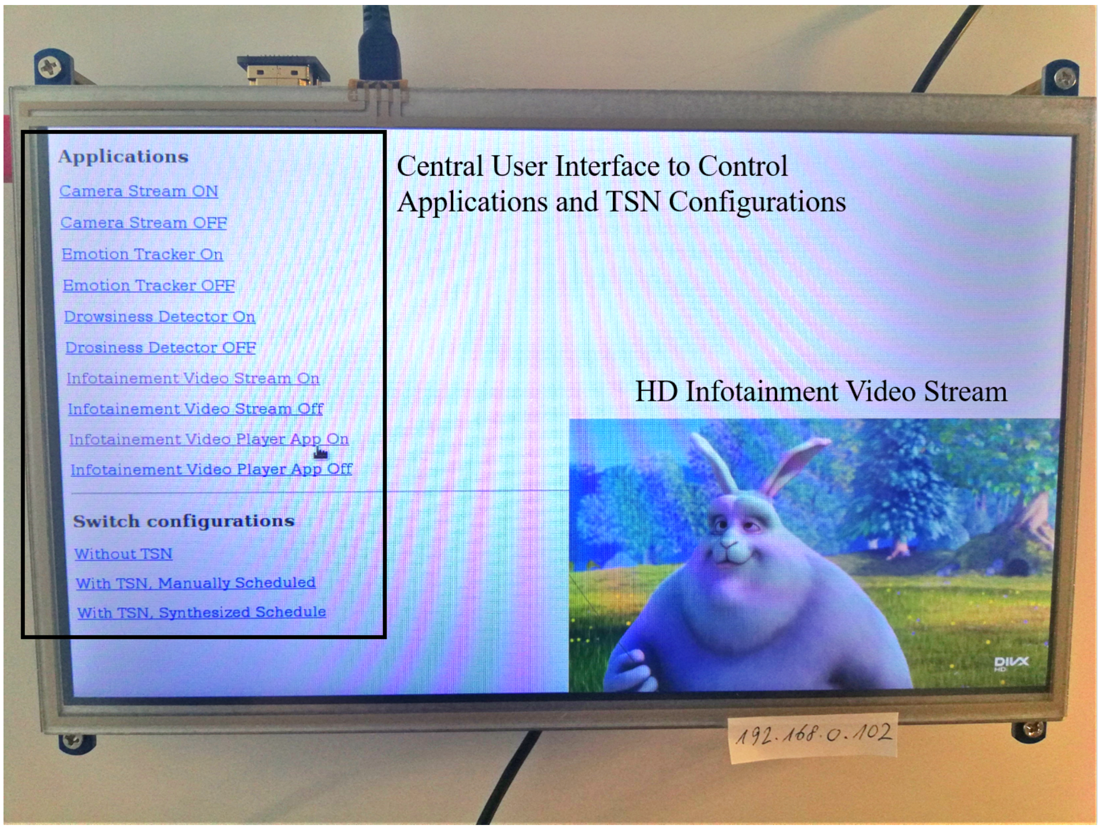

Applications
Camera Stream ON
Camera Stream OFF
Emotion Tracker On
Emotion Tracker OFF
Drowsiness Detector On
Drosiness Detector OFF
Infotainement Video Stream On
Infotainement Video Stream Off
Infotainement Video Player App On
Infotainement Video Player App Off
Switch configurations
Without TSN
With TSN, Manually Scheduled
With TSN, Synthesized Schedule
Central User Interface to Control Applications and TSN Configurations
HD Infotainment Video Stream
192.168.0.102

Figure 7.4: Central user interface to control applications.

It runs on a PC (cf. Figure 7.1) armed with an Intel i7 CPU (2.67 GHz) and 4 GB RAM. The software can be configured to generate an arbitrary network load up to 1 Gbit/s. The stream size and VLAN tags can be modified for different experiment scenarios. An excerpt of the software is depicted in Figure 7.3.

7.1.5 Central User Interface

A central web-based user interface is developed to start and stop the demo applications. It is also used to set up different TSN GCL configurations (it does not have anything to do with the modeling framework. It is only used for a basic GCL configuration). The user interface application is executed on node E3 which is a Raspberry Pi mini-computer (Pi 3 Model B) with an ARM Cortex-A53 CPU (1200 MHz) and 1GB RAM. The hardware provides a 100 Mbit/s Ethernet interface. Using the control application, the video streams can be turned on and off. The drowsiness application itself is also controlled by the user interface. Regarding the infotainment demo, the video file streamer (on E2) and player application (on E3) are also controlled by the interface. A screenshot of the user interface is depicted in Figure 7.4.

7.2 Initial Observations

Before starting with latency measurements, the setup is tested by starting the developed applications. In a first step, switches are configured with simple IEEE 802.1Q capability without any time-triggered features. First, the motor application (VLAN priority 6 and size of 200 Bytes) is started and the motors rotate synchronously as expected. Secondly, we start the webcam stream and the drowsiness application (VLAN priority 5). Both applications work without any disturbance. Thirdly, the infotainment application is started with multiple video streams (also VLAN priority 5) generating a maximum load of 100 Mbit/s traffic. Similar to the last two steps, the motor application works very well as expected. However, sometimes, a minimum acoustic disturbance can be perceived by the motor rotations indicating a minimum delay which is caused by the video transmission. Fourthly, we increase the network traffic (VLAN priority 5) to around 120 Mbit/s where the mentioned network load generator is used. The perceived acoustical and visual disturbance of the motor rotation is major while only 12% of the available bandwidth (1 Gbit/s) is used. In contrast to the motor application, drowsiness and infotainment applications are very stable when network load is increased. Artifacts in the videos can only be observed when we increase the network load to 1 Gbit/s with frames of priority 5 or higher.

To demonstrate how to protect critical stream with IEEE 802.1Qbv even in worst-case network load situations, we define a roughly estimated GCL table as depicted in Table 7.1. The first $20\mu s$ are blocked as a guard-band to avoid transmission of any streams which can have an interference with the step motor streams. The second GCL entry shows that $60\mu s$ are reserved to transmit the critical streams (conservatively estimated value) with the VLAN priority 6. The rest of the cycle is reserved for all other traffic except 6 and especially, we protect the SSH connections (priority 0) to the end-stations with the last entry.

To test the GCL configuration, disturbing traffic with the highest priority (7) is generated which tries to occupy all available 1 Gbit/s network bandwidth. It can be observed, that the motor application is executed without any misbehavior. For testing purposes, we modified entry 2 to 00000011 allowing the priority 7 traffic interfere with priority 6 (step motors). As a

Table 7.1: A roughly estimated GCL table for TSN ports of the switches (GCL cycle time is 10 ms which comes from the step motor application as mentioned). The transmission of critical step motor frames is protected against all other traffic. A guard-band in the beginning avoid any interferences.

Entry No.	Time Interval (μs)	Prio 0	Prio 1	Prio 2	Prio 3	Prio 4	Prio 5	Prio 6	Prio 7	Reserved for
1	20	0	0	0	0	0	0	0	0	Guard-band
2	60	0	0	0	0	0	0	1	0	Step motors
3	9900	1	1	1	1	1	1	0	1	All other traffic
4	20	1	0	0	0	0	0	0	0	SSH Connections

result, the step motors fail directly because of the too-long latencies ($> 6.3\ \mu s$). By modifying the entry to the original value (00000010), the step motors again rotate properly.

7.3 Model-based Configuration

Motivated by the introduced automotive demo applications, the developed framework is used to model parts of the experimental setup as depicted in Figure 7.5. We only focus on time-critical parts which are relevant for time-triggered scheduling and relevant for the next experiments. The rest of the applications (topics and streams) are scheduled for the best effort slot (similar to the streams between load generator and receiver) and do not require to be considered for the time-triggered scheduling.

To simulate more complex overlapping streams and challenge the framework, we add additional time-triggered topics (6) causing further competing streams. It simulates the existence of imaginary step motors which have to be considered for scheduling. The topics are published by two publishers *stepMotorPublisher_1* and *stepMotorPublisher_2* which are sent to the *stepMotorConsumer* end-station.

A GCL schedule visualizer is developed (Figure 7.6) where time-triggered slots (red), the guard-bands (black), and helper lines (blue) are presented to verify that: (i) the number of slots for each stream is equal to GCL hyper period divided by stream's period and (ii) there is no overlapped slots. Moreover, the analysis of the density of the computed schedule is possible. The density of the schedule is defined as how compact or distributed are the time-triggered slots positioned in the hyper period. It can have an important impact on fulfilling the deadline requirements of event-based sporadic streams.

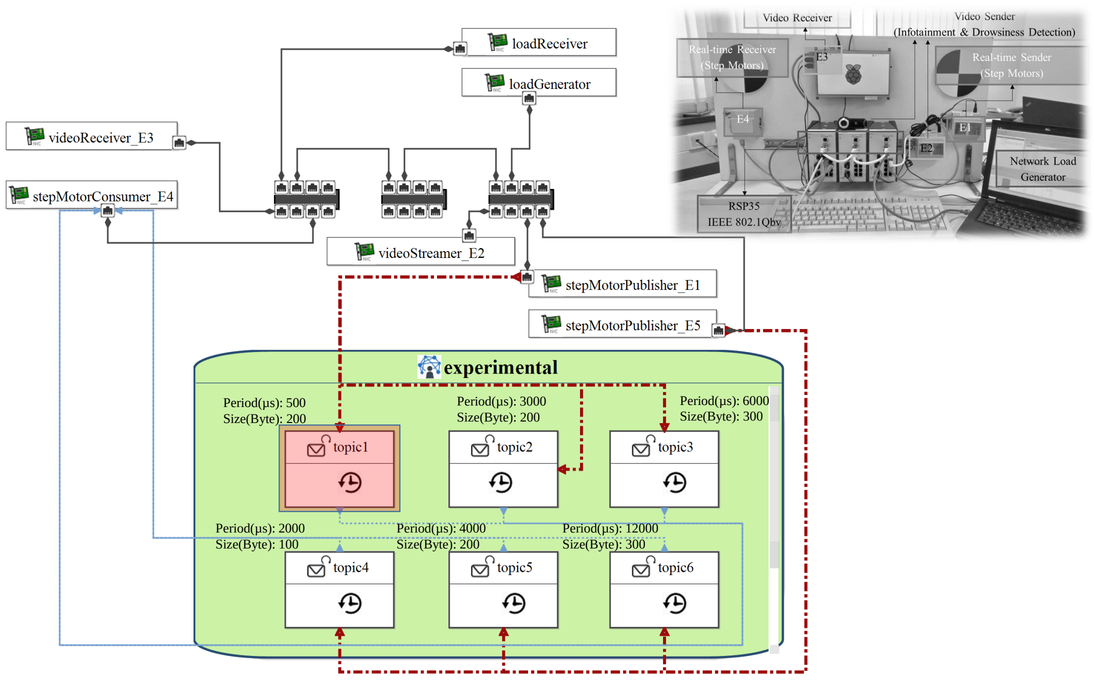


Figure 7.5: The network model of the experimental setup.

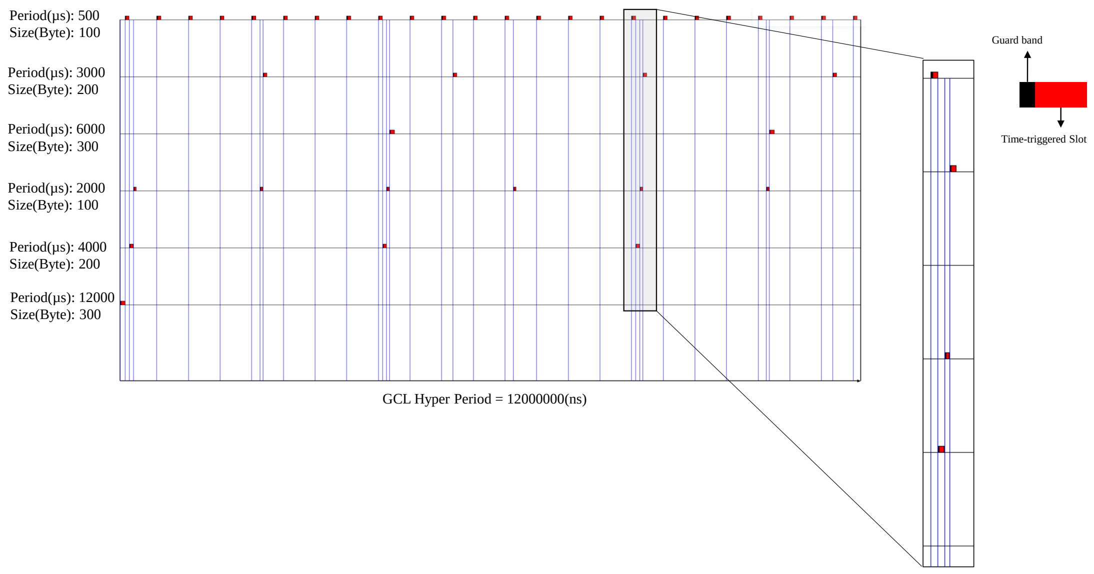


Figure 7.6: The implemented schedule visualizer depicts the position of the scheduled slots. Blue lines show that there are no overlapped slots. The space between red slots can be used to transmit best-effort traffic.

7.4 IEEE 802.1Qbv: Test Cases

The subject of the investigations here is periodic real-time frames in the network which will compete with the unscheduled traffic. The switches are configured fully automated by the framework.

Using hardware-based time stamping features of the switches (in range of few nanoseconds), the latency of the streams are measured with high precision. Because hardware-based timestamps are only available in the switches and not in the end-stations, we measure the latency of a stream as $t_{latency} = t_{out} - t_{in}$ where t_{in} is the arriving time of the stream in the first ingress switch port and t_{out} is the time when the stream leaves the last egress switch port towards the receiver end-station. This approach is used in all experiments to avoid measuring disturbance caused by the end-stations.

According to the specification, there is no time-triggered scheduling mechanism in IEEE 802.1Q. Hence, the expectation is that the frames with the highest priority are transmitted faster on average which reduces their experienced average latency and jitter.

Test Case 1. *Maximum and average latency of a real-time periodic stream, when applying 802.1Q, depends on the priority and network load generated by other competing frames.*

The missing time-triggered control mechanism in IEEE 802.1Q is introduced in IEEE 802.1Qbv. However, if there is no synchronization between the end-stations, the sender of a periodic stream will miss its slot with high probability and has to wait for its next slot. This phenomenon may increase the average latency and jitter compared to the situation when all nodes are synchronized. However, it guarantees transmission in the next slot which is an upper bound for the worst-case latency.

Test Case 2. *By applying IEEE 802.1Qbv without synchronization of the end-stations, a deterministic latency upper bound for a periodic time-triggered stream is guaranteed if and only if there are no other* unscheduled *frames with the same priority (higher or lower priorities are allowed).*

Synchronization in TSN is described in IEEE 802.1AS-Rev which exploits Precision Time Protocol (PTP) described in IEEE 1588-2008. The PTP functionality is implemented either software-based or hardware-based. A hardware-based time stamping leads to synchronization with higher precision. In spite of this advantage, it narrows down the choice of network designers regarding the embedded hardware and may cause additional hardware costs. This costs

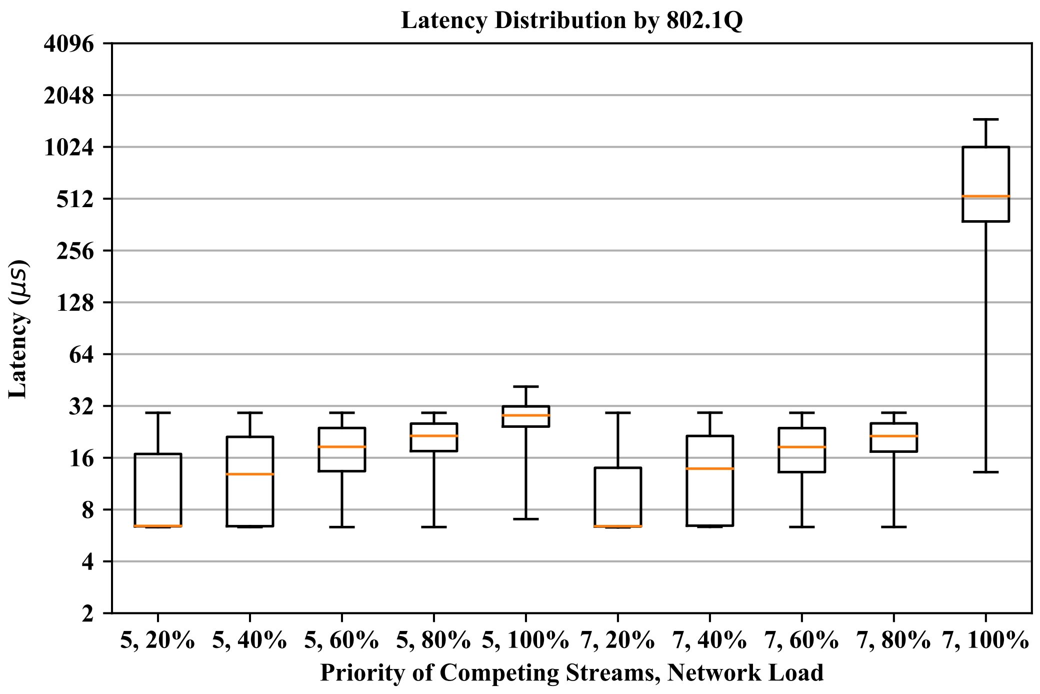


Figure 7.7: Higher traffic load and priority of the unscheduled traffic leads to higher latency and jitter for the time-triggered stream.

may be significant when network components are used in serial productions such as in the automotive domain. We intend to investigate the capabilities of the cheaper software-based PTP implementation.

Test Case 3. *IEEE 802.1Qbv with software-based PTP synchronization achieves the lowest latency and jitter values for a periodic stream compared to Test Case 1 if and only if all other competing streams with the same priority are scheduled and synchronized.*

Note: *The subject of investigation in the following experiments is the latency and jitter behavior of a* ***time-triggered periodic stream*** *with a period of* $500\mu s$*, a size of* 200 *Bytes, and an exclusive QoS priority* 6 *which competes with other traffic.*

7.4.1 Experiment 1: IEEE 802.1Q

In this experiment, we measure the timing impact of the priority and load of the unscheduled traffic on the time-critical stream to validate Test Case 1 where no time-triggered control mechanism is available. The latency is measured by increasing the unscheduled network load with data rates of $200, 400, 600, 800$ and 1080 Mbit/s (to create network congestion) once with lower priority 5 and once with higher priority 7.

The results as depicted in Figure 7.7 show, that increasing the network load leads to a higher maximum and average latency values. The impact is maximum if the unscheduled traffic has a higher priority than the critical stream and the network is congested (cf. $7, 100\%$). If all unscheduled stream has a priority less than the priority of the critical stream, lower average latencies can be achieved even if the network is congested (cf. $5, 100\%$). As expected, it can be observed that the achieved worst-case and average latency of periodic streams when using IEEE 802.1Q, depends on the shape of unscheduled network traffic.

7.4.2 Experiment 2: IEEE 802.1Qbv without synchronization

In this experiment, the gate controlling feature of 802.1Qbv is examined. The GCL computed based on the network model is used to configure time slots defining when and how long the gates of an egress port are open for a given QoS priority class. We use priority 5 and 7 and assign them to the unscheduled traffic and measure the worst-case latency for our critical stream with priority 6.

The cycle time of the GCL is set to $500\mu s$ equal to the period of the critical stream. We reserve the first $25\mu s$ of the cycle for the transmission of the critical stream exclusively and use the remaining slot for the unscheduled traffic. In this experiment, the end-stations are not synchronized. Thus, they measure the current time based on their CPU clock. In this experiment, 20000 samples of the critical stream are fired once competing with priority 5 and once competing with priority 7 traffic. The network is congested and there is a random waiting time between two consequent samples. The waiting time is required to assure varying sending times for the samples causing different latencies. Independent from the random sending time, the expectation is that the maximum measured latency has to be less than $500\mu s$ according to the specification.

The results are presented in Figure 7.8 and show that as expected in Test Case 2, the maximum latency is bounded by $500\mu s$. The latencies in this experiment are higher than the latencies from the last experiment when only IEEE 802.1Q is applied. This is because of the missing clock synchronization in the network. The sender end-station has a different time than the rest of the network. It cannot hit its reserved slot and misses it and has to wait for maximum one GCL cycle. However, in contrast to IEEE 802.1Q, the latency is bounded independent from the load and priority of the unscheduled traffic.

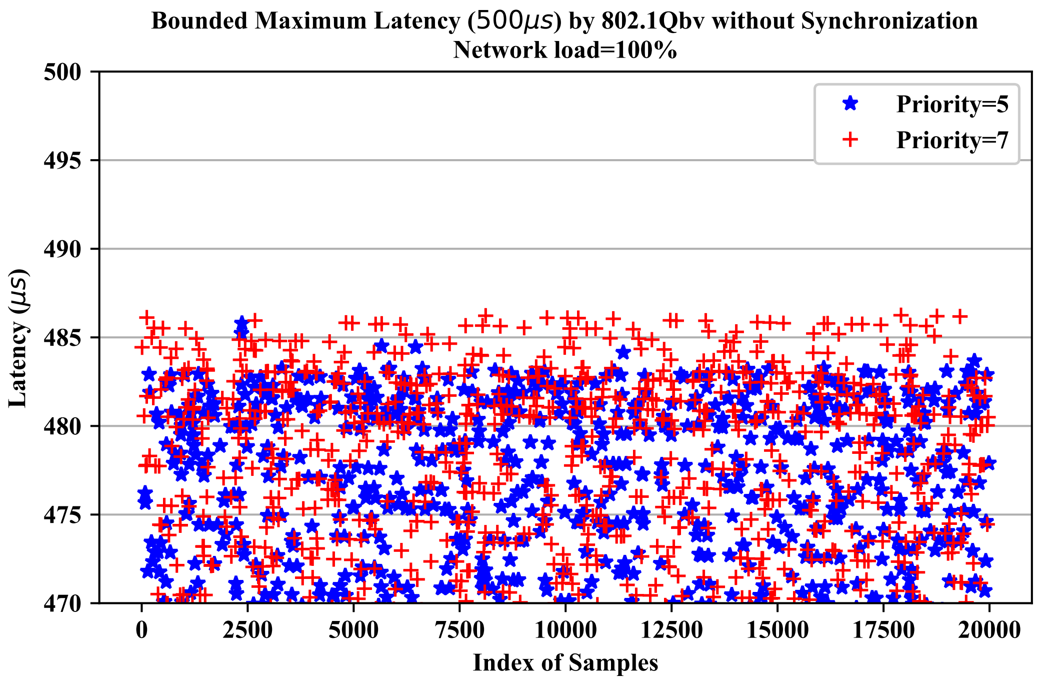


Figure 7.8: Independent from the priority of unscheduled traffic, the maximum latency of the time-triggered stream is guaranteed even when the network is congested.

The timing guarantee can be given if and only if there are no unscheduled streams with the priority 6 (a priority equal to the priority of the critical stream) in the network. Unscheduled or scheduled streams with the same priority will occupy the reserved time slot and increase the latency. Thus, to use IEEE 802.1Qbv without clock synchronization, formal timing analysis is required to verify if streams can fulfill their timing requirements [69].

Considering the measurements, IEEE 802.1Qbv can fulfill the requirements of latency-sensitive applications e. g. the before-mentioned drowsiness detection application with flexible jitter requirements.

7.4.3 Experiment 3: 802.1Qbv with PTP

In contrast to experiment 2, the end-stations are synchronized using a software-based implementation of PTP (Test Case 3). Similar to experiment 2, the assumption is that unscheduled traffic does not have the same priority as the critical streams. To configure the network the modeling framework is used as explained in Section 7.3. Based on the IEEE 802.1Qbv specifica-

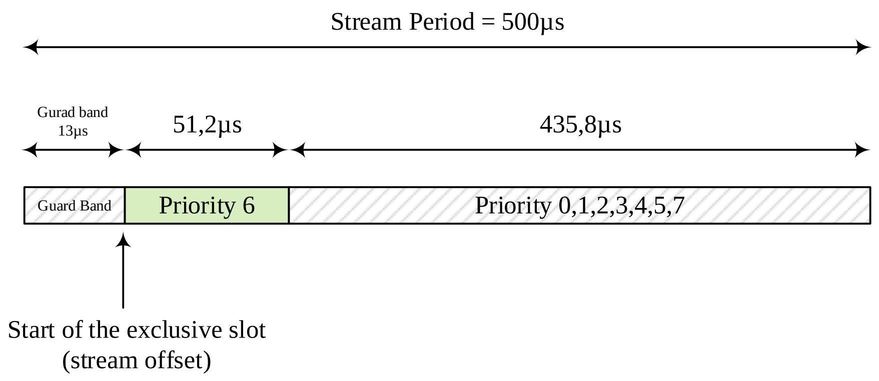


Figure 7.9: The computed schedule presents the configuration of the GCL for a single period of the time-triggered stream with priority 6.

tion, an end-station has to calculate its periodic sending time t_{send} to transmit stream instances correctly. It requires information about the *OperBaseTime* (t_{base}) which is the same value for all network nodes to calculate the starting time of the next GCL cycle, *OperCycleTime* (t_{cycle}) which is the cycle length and equal to the hyper period of all periodic streams traveling through an egress port, and its own *Offset* (t_{offset}) computed by the developed framework. It describes the exact position within a cycle reserved for the stream.

Figure 7.9 depicts the details of one period of the time-triggered stream. It is only a small part of the whole GCL cycle which is computed for this experiment (cf. Figure 7.6). The periodic stream has to arrive $13\mu s$ (Guard band) after the cycle starts. An important factor for correct timing calculation of t_{send} is the PTP offset values δ from slave to master. They are available in the statistics of the software-based PTP implementation. Based on measurements, we have $\delta_{median} = -561$ nanoseconds (Figure 7.10). The median value which has the highest expected value is selected for further calculation of the sending time. To calculate t_{send}, an end-station reads its own synchronized clock time $t_{current}$. Using the current time, it calculates the next sending time t_{send}:

$$\alpha = \lceil \frac{t_{current} - t_{base}}{t_{cycle}} \rceil$$
$$t_{send} = (\alpha * t_{cycle} + t_{offset}) - \delta_{median}$$

The results (Figure 7.11) show that the average latency and jitter are considerably lower in comparison to the previous experiments according to the expectations. Even if the network is overloaded and unscheduled streams have higher priority, IEEE 802.1Qbv scheduling mechanism

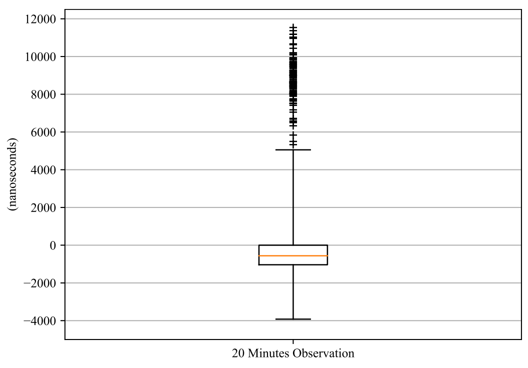


Figure 7.10: Observation of precision of the software-based PTP.

guarantees the highest degree of determinism. The measured values also contain the delay which is caused by the networking stack of the end-station. Without any conflicting traffic in the network, the measured latency for the critical stream is $\sim 6.3\mu s$. Values close to this have been expected because of IEEE 802.1Qbv control mechanism (no streams can compete with the critical stream because it has an exclusive slot). However, the results in Figure 7.11 show that the median of the latency is around $\sim 13\mu s$. The initial assumption is that this deviation is caused by a non-deterministic delay in the libpcap networking library of the Linux operating system. We plan to investigate this deviation in future work.

Although the end-stations are synchronized, in very rare cases (two extreme *outlier* values in 20000 cycles), the sender misses one GCL cycle. It leads to a worst-case latency equal to the cycle length ($500\mu s$). These latency outliers are caused by the inaccuracy of the PTP software which does not support hardware-based time stamping and the non-deterministic networking stack delay of the application on the end-stations. To avoid the problem of missing the period, two solutions can be proposed. A first conservative solution is to send the streams earlier than the planned t_{send} for example $48\mu s$ (Figure 7.11). It guarantees that the frames will be available in the egress queues of the corresponding ports for transmission when the gates open. This approach solves the problem of outliers, however, the streams' latency in average is increased to $\sim 31\mu s$ because they arrive too early (cf. Figures 7.11 and 7.12) and must wait longer. Based on the application logic and requirements, it can be decided if such this solution is suitable. The second solution is to arm the end-stations with a real-time operating system and hardware-based PTP support to have full control of the network stack and the hardware interrupts. Following this approach, outliers can be eliminated with respective additional hardware or other developing costs.

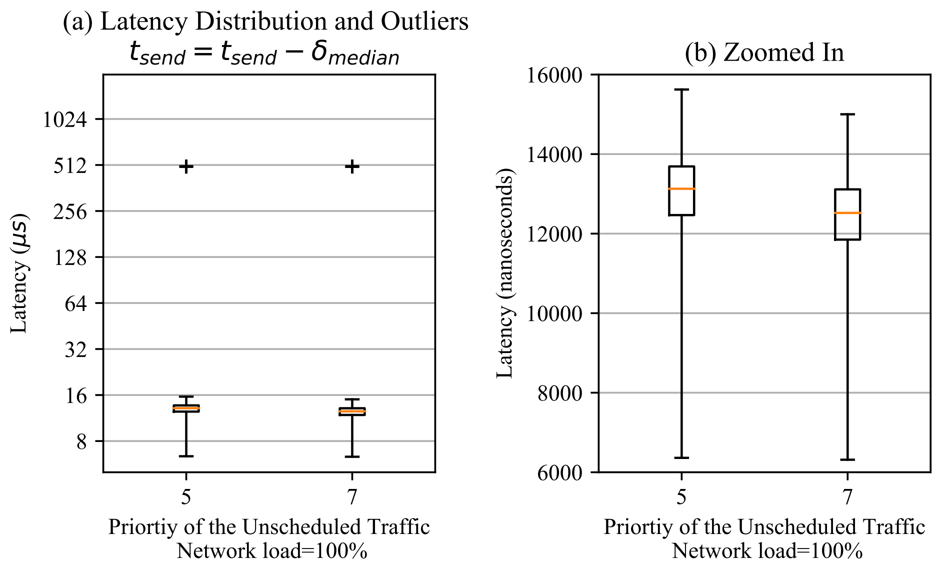


Figure 7.11: PTP-synchronized 802.1Qbv minimizes the latency values for the time-triggered stream. There are two extreme outliers from 20000 samples caused by not deterministic network stack of the sending end-station.

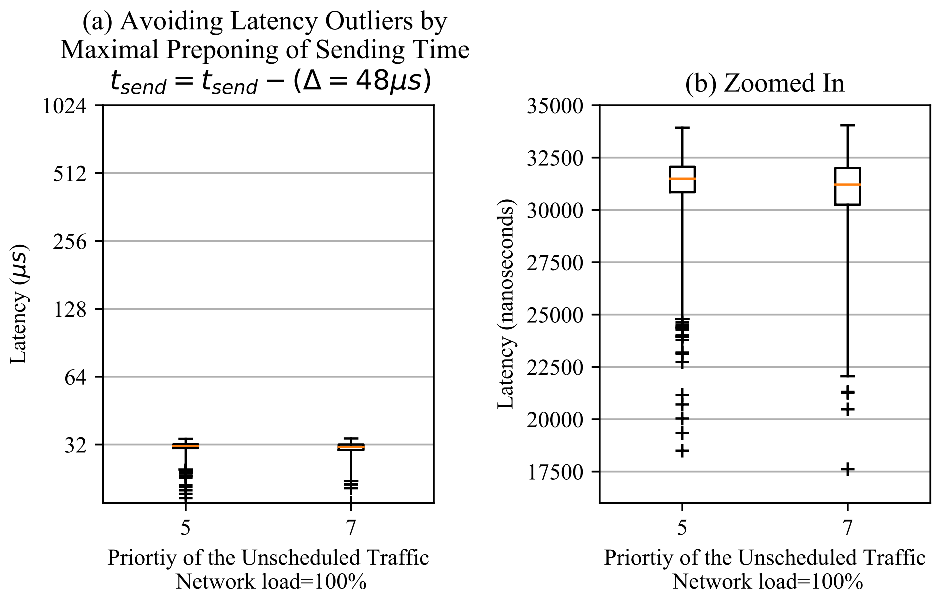


Figure 7.12: By increasing Δ, no stream misses its GCL slot and latency outliers remain below $32\mu s$.

Chapter 8

Conclusion and Future Work

8.1 Discussion of the Research Questions

The developed concepts in this thesis aim at answering three research questions dealing with (i) model-based automation of configuration procedures in TSN networks, (ii) backtracing of network design errors, and (iii) dynamic configuration at run-time. In the following, the achieved results for each research question are summarized.

How can the complexity of TSN schedule synthesis be reduced?

To automate the steps required for time-triggered schedule synthesis, a unique, graphical modeling approach is introduced and based on that, a framework is developed. It combines the strengths of the object-oriented modeling approach, the logic programming paradigm, and Satisfiability Modulo Theories. Network designers can use the developed modeling framework to create a graphical network model including information about hardware, software, and time requirements. The information is used by the framework to compute correct schedules and to configure the network nodes (GCL tables of egress ports) automatically. The implementation details of the framework and a modeling example is presented. However, it is also possible that network design errors cause model infeasibility. Moreover, inefficient models can significantly increase the synthesis time which decelerates application development.

How to facilitate model verification before schedule synthesis and support infeasibility analysis after synthesis?

To verify the infrastructural requirements for networking based on TSN standards, adequate inference rules and queries are developed. These Prolog-based rules are used to avoid network configurations which lead to long scheduling synthesis time. For back-tracing of infeasible network models, a constraint labeling approach is developed. While building the scheduling constraints for IEEE 802.1Qbv time-triggered streams, the constraints are labeled with additional information about the streams. If the solving procedure fails, it produces an unsatisfiable core containing conflicting constraints. The labels of these constraints assist in finding those network fractions where the design error(s) must be corrected. The iterative developed approach is demonstrated by an exemplary complex network model consisting of four application domains.

How to facilitate autoconfiguration of TSN networks with a focus on IEEE 802.1Qbv standard?

A data-centric-based approach is proposed to enable the TSN networks for dynamic configuration at run-time. The proposed approach acts as a bridge between the networking (MAC) and middleware/application layer. The middleware layer provides dynamic networking services (such as device discovery and the transmission of data between publishers and subscribers) and the underlying TSN mechanisms such as IEEE 802.1Qbv guarantee the tight timing of the application streams by the adequate configuration of the egress ports. Any new changes in the application layer are communicated to the developed model-based synthesis approach. It decides if re-scheduling of the network is required. In this case, the middleware waits for the required synthesis steps until the configuration at TSN layer is successfully finished.

8.2 Limitations

The developed framework is the first step towards fully-automated configuration and verification of TSN networks. In the following, the limitations of the framework are discussed.

- **Formulation of adequate Prolog queries**

  In this thesis, network facts are automatically generated from the graphical model. A set of rules and queries is developed to handle verification and configuration tasks. However, the developed rules and queries do not cover all possible networking use-cases and scenarios. Formulation of correct Prolog queries for specific configuration and verification tasks (which considers all available TSN standards including their details) requires a background in logic programming. Moreover, the current implementation only allows for generating network knowledge base from a graphically created network model, but not the other way around. The implementation of a model generation in both directions is required to increase the automation level of model capturing and modifications at run-time.

- **Software verification**

  The developed framework is modular and exploits existing software artifacts such as Z3 solver, Eclipse modeling framework, etc. These software modules are used for the synthesis of the schedule of hard, real-time streams which may have safety relevance. The quality and correctness of the synthesis results depend highly on the correct implementation of these software modules. Verification of the software components has a direct impact on the certification of the safety-relevant parts of the network which are configured using the framework.

- **Understanding of unsatisfiability**

  The presented constraint-labeling approach for back-tracing and iterative correction of the design errors contains only the information which aids in localizing the erroneous network segments. However, it does not produce any explanations or correcting recommendations. There is also no guarantee of how many iterations are required to achieve satisfiability of the network model. The number of iterations depends on the correction actions and modifications of the model. Incorrect modifications can also cause more conflicting constraints. Thus, networking expertise and application understanding are required to derive adequate modifications in the topology or application logic.

8.3 Future Work

For future work, experts and users from the field of industrial networking will evaluate the features of the developed graphical modeling tool. Based on the results, challenges for future improvements of the framework can be identified and investigated. In particular, alternative visualizations of the model elements can be investigated and evaluated by adequate surveys.

Developing of configuration constraints is an ongoing topic within the TSN communities and different scheduling, routing, and constraint solving approaches are expected to appear soon. Considering this, one significant challenge is the long synthesis times for time-triggered streams. This issue is tackled with an assisting tool for better network management. However, this solution is more suitable for flexible network topologies (e. g. fewer limitations on the number of available switches) such as in factories. For very compact network topologies with a very low number of TSN switches and a high number of critical streams (e. g. in-vehicle networks), the problem of long synthesis time remains challenging. Future work will investigate heuristic-based approaches to mitigate this problem which has a significant impact on very dynamic networks.

With a valid TSN configuration (e. g. correctly synthesized time-triggered schedules), the desired behavior is guaranteed based on underlying mathematical backgrounds such as logic-programming. However, if the network nodes have deviations in their expected behavior, the impact on the whole system will be unknown. This challenge is not negligible concerning safety-related applications. In future work, different approaches including Fault Tree Analysis (FTA) will be investigated for their feasibility to tackle this problem. The results are expected to flow into the framework to verify for example system safety for the probability of deviation in node behavior.

8.4 Industrial Practice

The features of the developed framework are unique in the automated configuration of TSN networks. The implemented functionality offers an initial tool which can be applied and tested by switch manufacturers, network designers, and application developers in the automation industry. These developed novel features could also be extended based on a prioritization of industry requirements in both configuration and infrastructural verification directions.

In addition to the TSN use-case, the logic-based modeling approach and the constraint-labeling methodology presented in this thesis could also be applied to many other diverse application fields. Any system that can be described in mathematical logic (facts) and is required to be verified against specifications (rules and queries) is a potential use-case. This modeling tool automatically generates the required knowledge base (Prolog facts) for desired applications and, enables an analysis of design errors.

Appendix A

Appendix

```
switch(sw1, [sw1_p1, sw1_p2, sw1_p3, sw1_p4, sw1_p5, sw1_p6, sw1_p7, sw1_p8]).
switch(sw2, [sw2_p1, sw2_p2, sw2_p3, sw2_p4, sw2_p5, sw2_p6, sw2_p7, sw2_p8]).
switch(sw3, [sw3_p1, sw3_p2, sw3_p3, sw3_p4, sw3_p5, sw3_p6, sw3_p7, sw3_p8]).
domain(alert_domain).
estation(display_node, [es3_p1]).
estation(monitoring_node, [es4_p1]).
estation(network_load_generator, [es4_p1]).
isLinked(sw1_p1, sw2_p2, 1000000000).
isLinked(sw2_p1, sw3_p1, 1000000000).
isLinked(es3_p1, sw3_p5, 1000000000).
isLinked(es4_p1, sw3_p7, 1000000000).
isLinked(es4_p1, sw1_p2, 1000000000).
topic(alert_domain, alert_domain_system_status, periodic, 20000000, 800).
qosLatency(alert_domain_system_status_q1, alert_domain, alert_domain_system_status,
   ↪ 40000000, 7).
publishes(monitoring_node, es4_p1, alert_domain, alert_domain_system_status).
consumes(display_node, es3_p1, alert_domain, alert_domain_system_status).
```

Prolog A.1: NKB before *critical_node* joins the network.

```
estation(critical_node, [es2_p1]).
isLinked(es2_p1, sw1_p5, 1000000000).
topic(alert_domain, alert_domain_cpu_load, periodic, 1000000, 200).
topic(alert_domain, alert_domain_freemem, periodic, 20000000, 200).
topic(alert_domain, alert_domain_disk_status, periodic, 100000000, 200).
topic(alert_domain, alert_domain_cpu_temperature, periodic, 1000000, 200).
qosLatency(alert_domain_cpu_load_q1, alert_domain, alert_domain_cpu_load, 2000000, 7).
qosLatency(alert_domain_freemem_q1, alert_domain, alert_domain_freemem, 40000000, 7).
qosLatency(alert_domain_disk_status_q1, alert_domain, alert_domain_disk_status,
    ↪ 200000000, 7).
qosLatency(alert_domain_cpu_temperature_q1, alert_domain, alert_domain_cpu_temperature
    ↪ , 2000000, 7).
publishes(critical_node, es2_p1, alert_domain, alert_domain_cpu_load).
publishes(critical_node, es2_p1, alert_domain, alert_domain_freemem).
publishes(critical_node, es2_p1, alert_domain, alert_domain_disk_status).
publishes(critical_node, es2_p1, alert_domain, alert_domain_cpu_temperature).
consumes(monitoring_node, es4_p1, alert_domain, alert_domain_cpu_load).
consumes(monitoring_node, es4_p1, alert_domain, alert_domain_freemem).
consumes(monitoring_node, es4_p1, alert_domain, alert_domain_cpu_temperature).
```

Prolog A.2: NKB after the *critical_node* joins the network.

```
topic(applicationDomain1, sensor1, periodic, 250000, 400).
topic(applicationDomain1, sensor2, periodic, 125000, 10).
topic(applicationDomain1, sensor3, periodic, 500000, 200).
topic(applicationDomain1, sensor4, sporadic, 10000000, 1500).
topic(applicationDomain1, sensor5, periodic, 2000000, 6000).
topic(applicationDomain1, sensor6, periodic, 125000, 40).
topic(applicationDomain1, actuator1, periodic, 500000, 200).
topic(applicationDomain1, actuator2, periodic, 500000, 150).
topic(applicationDomain1, actuator3, periodic, 500000, 300).
topic(applicationDoamin2, sensor1, periodic, 250000, 200).
topic(applicationDoamin2, sensor2, periodic, 500000, 200).
topic(applicationDoamin2, sensor3, periodic, 500000, 200).
topic(applicationDoamin2, sensor4, periodic, 2000000, 500).
topic(applicationDoamin2, sensor5, periodic, 5000000, 1200).
topic(applicationDoamin2, sensor6, periodic, 10000000, 500).
topic(applicationDoamin2, sensor7, periodic, 20000000, 1400).
topic(applicationDoamin2, actuator1, periodic, 250000, 200).
topic(applicationDoamin2, actuator2, periodic, 500000, 500).
topic(applicationDomain3, sensor1, periodic, 250000, 200).
topic(applicationDomain3, sensor2, periodic, 500000, 100).
topic(applicationDomain3, sensor3, periodic, 10000000, 150).
topic(applicationDomain3, actuator1, periodic, 250000, 200).
topic(applicationDomain3, actuator2, periodic, 500000, 400).
topic(applicationDomain3, actuator3, periodic, 10000000, 200).
topic(applicationDomain4, sensor1, periodic, 125000, 40).
topic(applicationDomain4, sensor2, periodic, 1000000, 5000).
topic(applicationDomain4, sensor3, periodic, 10000000, 500).
topic(applicationDomain4, actuator2, periodic, 250000, 50).
```

Prolog A.3: Model correction example including stream period (*ns*) and size (*Byte*).

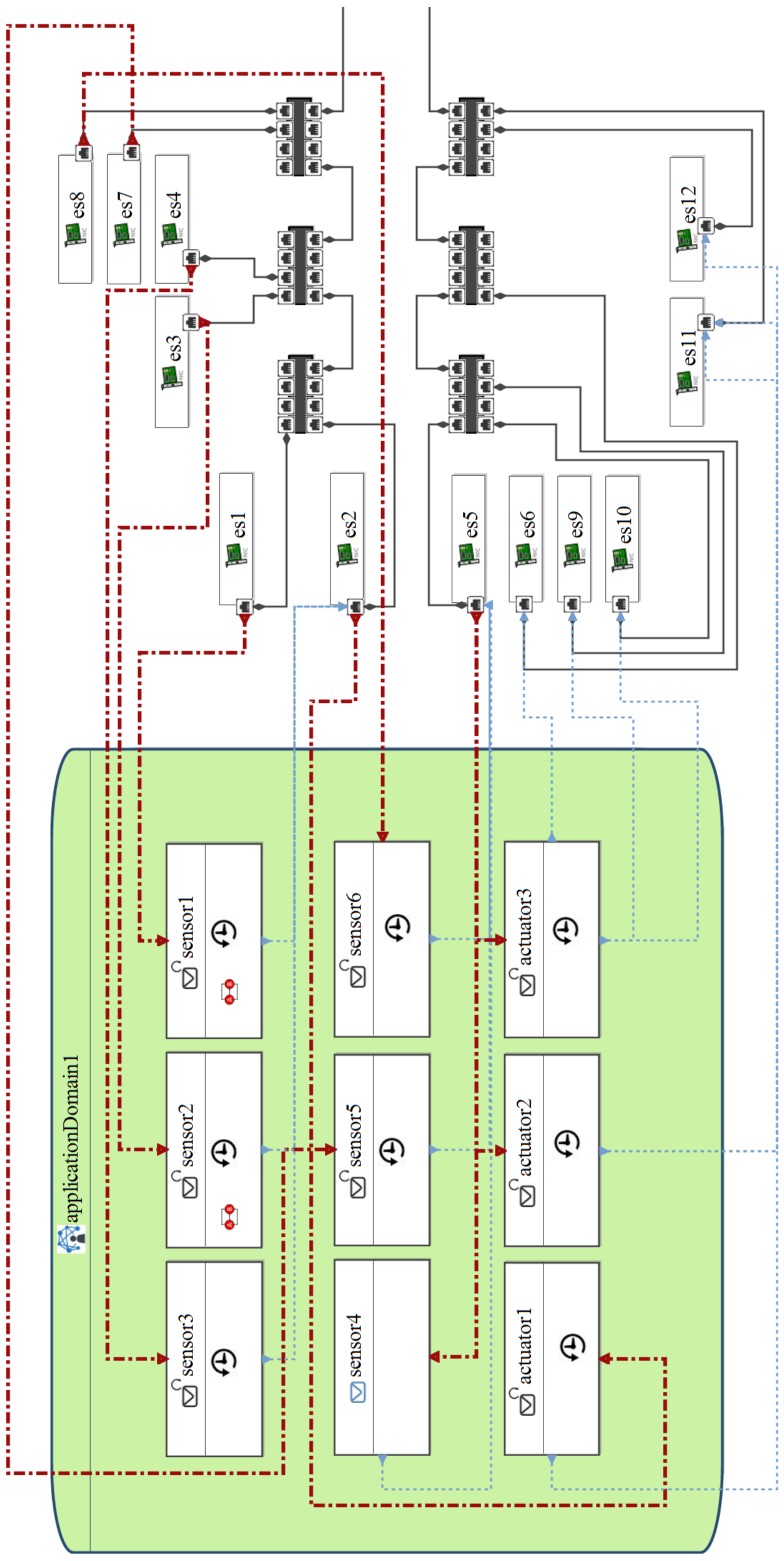


Figure A.1: Domain application 1 in details.

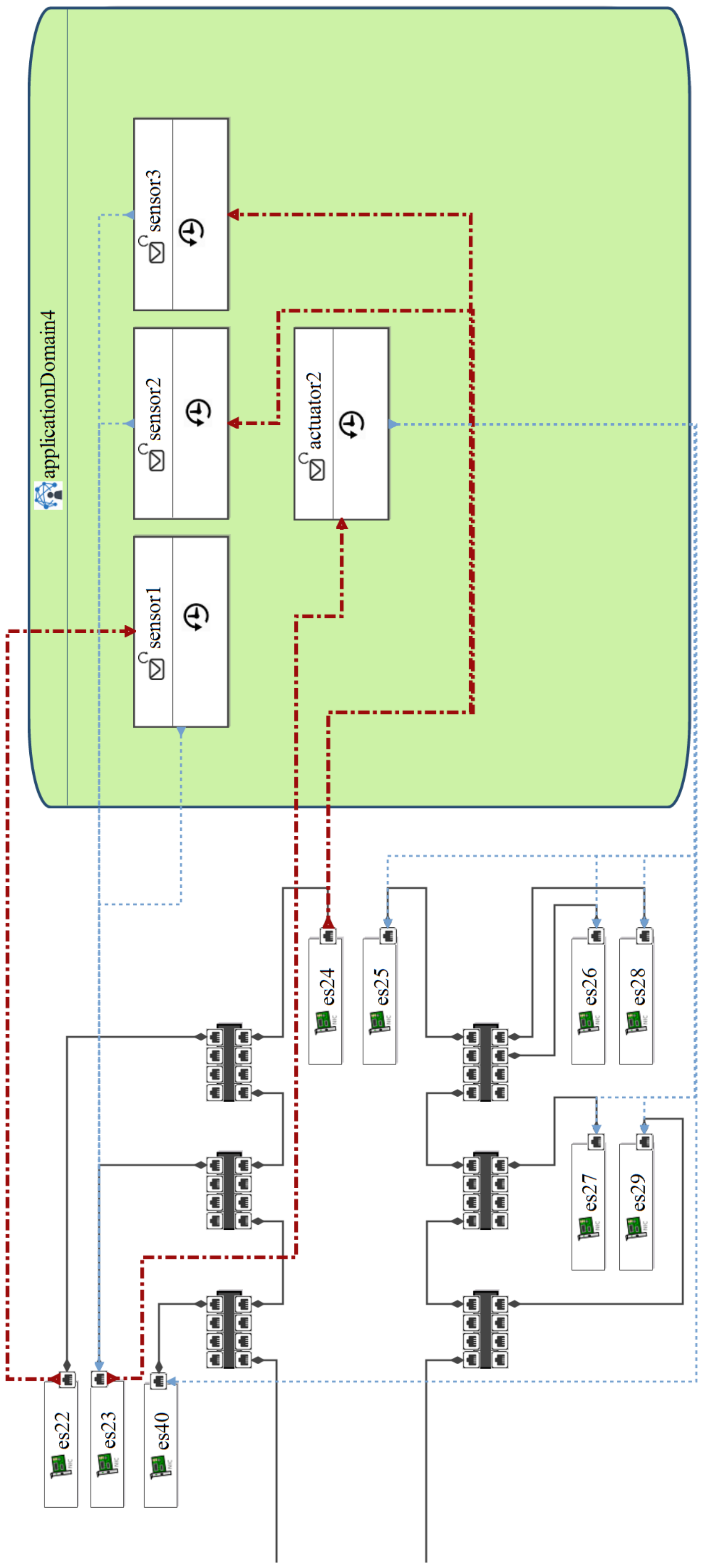


Figure A.2: Domain application 4 in details.

References

[1] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. **The time-triggered ethernet (TTE) design**. In *Object-Oriented Real-Time Distributed Computing, 2005. ISORC 2005. Eighth IEEE International Symposium on*, pages 22–33. IEEE, 2005. 1

[2] **Time-triggered Ethernet**. http://standards.sae.org/as6802/. Accessed: 22.01.2018. 1, 19

[3] Dirk Jansen and Holger Buttner. **Real-time Ethernet: the EtherCAT solution**. *Computing and Control Engineering*, **15**(1):16–21, 2004. 1

[4] **Open Powerlink - An Open Source Powerlink protocol stack**. http://openpowerlink.sourceforge.net/. Accessed: 22.01.2018. 1

[5] Morteza Hashemi Farzaneh, Suraj Nair, Mohammad Ali Nasseri, and Alois Knoll. **Reducing communication-related complexity in heterogeneous networked medical systems considering non-functional requirements**. In *Advanced Communication Technology (ICACT), 2014 16th International Conference on*, pages 547–552. IEEE, 2014. 1

[6] **Audio Video Bridging**. http://www.ieee802.org/. Accessed: 14.06.2016. 1

[7] **IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams**. *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pages C1–72, Jan 2009. 2, 7

[8] **IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)**.

IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005), pages 1–119, Sept 2010. 2

[9] **IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks**. *IEEE Std 802.1AS-2011*, pages 1–292, March 2011. 2

[10] **Time-Sensitive Networking**. http://www.ieee802.org/. Accessed 10.04.2017. 2

[11] **IEEE Standard for Local and metropolitan area networks– Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks– Corrigendum 2: Technical and Editorial Corrections**. *IEEE Std 802.1AS-2011/Cor 2-2015 (Corrigendum to IEEE Std 802.1AS-2011)*, pages 1–13, April 2016. 3

[12] **Time Sensitive Networking: Deep Impact oder Mission Impossible?** https://www.pc-control.net. Accessed: 22.01.2018. 3

[13] **IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic**. *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pages 1–57, March 2016. 3

[14] **IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption**. *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)*, pages 1–52, Aug 2016. 3

[15] **IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 29: Cyclic Queuing and Forwarding**. *IEEE 802.1Qch-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd(TM)-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, IEEE Std 802.1Qbz-2016, and IEEE Std 802.1Qci-2017)*, pages 1–30, June 2017. 3

[16] **Bridges and Bridged Networks Amendment: Asynchronous Traffic Shapingl**. https://1.ieee802.org/tsn/802-1qcr/. Accessed: 22.01.2018. 4

[17] **IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability**. *IEEE Std 802.1CB-2017*, pages 1–102, Oct 2017. 4

[18] **IEEE Standard for Local and metropolitan area networks– Bridges and Bridged Networks - Amendment 24: Path Control and Reservation**. *IEEE Std 802.1Qca-2015 (Amendment to IEEE Std 802.1Q— as amended by IEEE Std 802.1Qcd-2015 and IEEE Std 802.1Q—/Cor 1-2015)*, pages 1–120, March 2016. 4

[19] Erich Bockard. **Everything You Need To Know About TSN Sub-Standards**. http://blog.ebv.com. Accessed: 26.12.2017. 4

[20] **IEEE Draft Standard for Local and metropolitan area networks–Media Access Control (MAC) Bridges and Virtual Bridged Local Area Networks Amendment: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements**. *IEEE P802.1Qcc/D2.0, October 2017*, pages 1–207, Jan 2017. 4, 64

[21] **Bridges and Bridged Networks Amendment: YANG Data Model**. https://1.ieee802.org/tsn/802-1qcp/. Accessed: 22.01.2018. 4

[22] **Bridges and Bridged Networks Amendment: YANG Data Model**. Automatic Attachment to Provider Backbone Bridging (PBB) services. Accessed: 22.01.2018. 4

[23] **IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing**. *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)*, pages 1–65, Sept 2017. 4

[24] **IEEE Draft Time-Sensitive Networking for Fronthaul**. *IEEE P802.1CM/D2.0, January 2018*, pages 1–72, Jan 2018. 4

[25] Wilfried Steiner. **Synthesis of static communication schedules for mixed-criticality systems**. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*, pages 11–18. IEEE, 2011. 13, 19, 34, 68

[26] Jan Dvořák, Martin Heller, and Zdeněk Hanzálek. **Makespan minimization of Time-Triggered traffic on a TTEthernet network**. In *Factory Communication Systems (WFCS), 2017 IEEE 13th International Workshop on*, pages 1–10. IEEE, 2017. 13, 32

[27] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty. **Task- and network-level schedule co-synthesis of Ethernet-based time-triggered systems**. In *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 119–124, 2014. 14, 34, 53, 54, 55, 61

[28] W. Steiner. **An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks**. In *31st IEEE Real-Time Systems Symposium*, pages 375–384, 2010. 14, 16, 34, 53, 55, 58, 61

[29] Silviu S. Craciunas and Ramon Serna Oliver. **Combined task- and network-level scheduling for distributed time-triggered systems**. *Real-Time Systems*, **52**(2):161–200, 2016. 14, 33

[30] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. **Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks**. In *Proc. of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 183–192. ACM, 2016. 14, 33, 53, 55, 90

[31] Christian Schöler, René Krenz-Bååth, Ayman Murshed, and Roman Obermaisser. **Computing optimal communication schedules for time-triggered networks using an SMT solver**. In *Industrial Embedded Systems (SIES), 2016 11th IEEE Symposium on*, pages 1–9. IEEE, 2016. 14, 33

[32] Heyuan Shi, Kun Tang, Chengbao Liu, Xiaoyu Song, Chao Hu, and Jiaguang Sun. **Memetic-based schedule synthesis for communication on time-triggered embedded systems**. *International Journal of Distributed Sensor Networks*, **13**(10):1550147717738167, 2017. 14, 33

[33] Francisco Manuel Pozo Pérez, Guillermo Rodriguez-Navas, Hans Hansson, and Wilfried Steiner. **Schedule Synthesis for Next Generation Time-Triggered Networks**, 2017. 14, 33

[34] Sune Mølgaard Laursen, Paul Pop, and Wilfried Steiner. **Routing optimization of AVB streams in TSN networks**. *ACM SIGBED Review*, **13**(4):43–48, 2016. 15, 33

[35] Fedor Smirnov, Michael Glass, Felix Reimann, and Jürgen Teich. **Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks**. In *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*, pages 1–6. IEEE, 2017. 15, 33

[36] Paul Pop, Michael Lander Raagaard, Silviu S Craciunas, and Wilfried Steiner. **Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks**. *IET Cyber-Physical Systems: Theory & Applications*, **1**(1):86–94, 2016. 15, 33

[37] Rouhollah Mahfuzi, Amir Aminifar, Soheil Samii, Ahmed Rezine, Petru Eles, and Zebo Peng. **Stability-Aware Integrated Routing and Scheduling for Control Applications in Ethernet Networks**. In *Design, Automation and Test in Europe (DATE)*, number EPFL-CONF-232889, 2018. 15, 32

[38] Naresh Ganesh Nayak, Frank Duerr, and Kurt Rothermel. **Routing Algorithms for IEEE802. 1Qbv Networks**. In *RTN workshop, ECRTS*, 2017. 15, 33

[39] Alexander Metzner, Martin Franzle, Christian Herde, and Ingo Stierand. **Scheduling distributed real-time systems by satisfiability checking**. In *Embedded and Real-Time Computing Systems and Applications, 2005. Proceedings. 11th IEEE International Conference on*, pages 409–415. IEEE, 2005. 15, 34

[40] Paul Pop, Petru Eles, and Zebo Peng. **An improved scheduling technique for time-triggered embedded systems**. In *EUROMICRO Conference, 1999. Proceedings. 25th*, **1**, pages 303–310. IEEE, 1999. 16, 35

[41] Paul Pop, Petru Eles, and Zebo Peng. **Schedulability-driven communication synthesis for time triggered embedded systems**. *Real-Time Systems*, **26**(3):297–325, 2004. 16, 35

[42] Zdenek Hanzálek, Pavel Burget, and Premysl Šucha. **Profinet IO IRT message scheduling**. In *Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on*, pages 57–65. IEEE, 2009. 16, 34

[43] **Everything You Need To Know About TSN Sub-Standards**. https://www.siemens.com. Accessed: 30.12.2017. 16

[44] Haibo Zeng, Wei Zheng, Marco Di Natale, Arkadeb Ghosal, Paolo Giusto, and Alberto Sangiovanni-Vincentelli. **Scheduling the FlexRay bus using optimization techniques**. In *Proceedings of the 46th Annual Design Automation Conference*, pages 874–877. ACM, 2009. 16, 34

[45] FlexRay Consortium et al. **FlexRay communications system-protocol specification**. *Version*, **2**(1):198–207, 2005. 16

[46] Florian Sagstetter, Sidharta Andalam, Peter Waszecki, Martin Lukasiewycz, Hauke Stähle, Samarjit Chakraborty, and Alois Knoll. **Schedule integration framework for time-triggered automotive architectures**. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014. 16, 31, 34

[47] Florian Sagstetter. *Schedule Synthesis for Time-Triggered Automotive Architectures*. PhD thesis, Universitätsbibliothek der TU München, 2016. 16, 31, 33

[48] **INET Framework**. https://inet.omnetpp.org/. Accessed: 02.01.2018. 17

[49] Till Steinbach, Hermand Dieumo Kenfack, Franz Korf, and Thomas C Schmidt. **An extension of the OMNeT++ INET framework for simulating real-time ethernet with high accuracy**. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, pages 375–382. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011. 17, 34

[50] Hyung-Taek Lim, Daniel Herrscher, and Firas Chaari. **Performance comparison of ieee 802.1 q and ieee 802.1 avb in an ethernet-based in-vehicle network**. In *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*, **1**, pages 1–6. IEEE, 2012. 17, 34

[51] Giuliana Alderisi, Alfio Caltabiano, Giancarlo Vasta, Giancarlo Iannizzotto, Till Steinbach, and Lucia Lo Bello. **Simulative assessments of IEEE 802.1 Ethernet AVB and time-triggered Ethernet for advanced driver assistance systems and in-car infotainment**. In *Vehicular Networking Conference (VNC), 2012 IEEE*, pages 187–194. IEEE, 2012. 17, 34

[52] Till Steinbach, Hyung-Taek Lim, Franz Korf, Thomas C Schmidt, Daniel Herrscher, and Adam Wolisz. **Tomorrow's in-car interconnect? A competitive evaluation of IEEE 802.1 AVB and Time-Triggered Ethernet (AS6802)**. In *Vehicular Technology Conference (VTC Fall), 2012 IEEE*, pages 1–5. IEEE, 2012. 17, 34

[53] Philipp Meyer, Till Steinbach, Franz Korf, and Thomas C Schmidt. **Extending IEEE 802.1 AVB with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic**. In *Vehicular Networking Conference (VNC), 2013 IEEE*, pages 47–54. IEEE, 2013. 17, 34

[54] Till Steinbach, Hyung-Taek Lim, Franz Korf, Thomas C Schmidt, Daniel Herrscher, and Adam Wolisz. **Beware of the hidden! How cross-traffic affects quality assurances of competing real-time Ethernet standards for in-car communication**. In *Local Computer Networks (LCN), 2015 IEEE 40th Conference on*, pages 1–9. IEEE, 2015. 17, 33

[55] Jaewoong Ko, Ju-ho Lee, Chulsun Park, and Sung-kwon Park. **Research on optimal bandwidth allocation for the scheduled traffic in IEEE 802.1 AVB**. In *Vehicular Electronics and Safety (ICVES), 2015 IEEE International Conference on*, pages 31–35. IEEE, 2015. 18, 33

[56] Peter Heise, Fabien Geyer, and Roman Obermaisser. **TSimNet: An Industrial Time Sensitive Networking Simulation Framework Based on OMNeT++**. In *New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on*, pages 1–5. IEEE, 2016. 18, 33

[57] Jonas Diemer, Daniel Thiele, and Rolf Ernst. **Formal worst-case timing analysis of Ethernet topologies with strict-priority and AVB switching**. In *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pages 1–10. IEEE, 2012. 18, 34

[58] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. **System level performance analysis - the SymTA/S approach**. *IEEE Proc. - Computers and Digital Techniques*, **152**(2):148–166, 2005. 18

[59] Rene Queck. **Analysis of ethernet avb for automotive networks using network calculus**. In *Vehicular Electronics and Safety (ICVES), 2012 IEEE International Conference on*, pages 61–67. IEEE, 2012. 18, 34

[60] Felix Reimann, Sebastian Graf, Fabian Streit, Michael Glass, and Jürgen Teich. **Timing analysis of Ethernet AVB-based automotive E/E architectures**. In *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*, pages 1–8. IEEE, 2013. 18, 34

[61] L. Thiele, S. Chakraborty, and M. Naedele. **Real-time calculus for scheduling hard real-time systems**. In *Circuits and Systems, 2000. Proc.. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, **4**, pages 101–104 vol.4, 2000. 18, 21, 26

[62] Unmesh D Bordoloi, Amir Aminifar, Petru Eles, and Zebo Peng. **Schedulability analysis of Ethernet AVB switches**. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on*, pages 1–10. IEEE, 2014. 18, 34

[63] Philip Axer, Daniel Thiele, Rolf Ernst, and Jonas Diemer. **Exploiting shaper context to improve performance bounds of ethernet avb networks**. In *Proceedings of the 51st Annual Design Automation Conference*, pages 1–6. ACM, 2014. 18, 34

[64] Jingyue Cao, Pieter JL Cuijpers, Reinder J Bril, and Johan J Lukkien. **Tight worst-case response-time analysis for ethernet AVB using eligible intervals**. In *Factory Communication Systems (WFCS), 2016 IEEE World Conference on*, pages 1–8. IEEE, 2016. 19, 33

[65] LuXi Zhao, Huagang Xiong, Zhong Zheng, and Qiao Li. **Improving worst-case latency analysis for rate-constrained traffic in the time-triggered ethernet network**. *IEEE Communications Letters*, **18**(11):1927–1930, 2014. 19, 34

[66] Domitian TamasSelicean, Paul Pop, and Wilfried Steiner. **Timing analysis of rate constrained traffic for the TTEthernet communication protocol**. In *Real-Time Distributed Computing (ISORC), 2015 IEEE 18th International Symposium on*, pages 119–126. IEEE, 2015. 19, 21, 33

[67] Traian Pop, Paul Pop, Petru Eles, and Zebo Peng. **Analysis and optimisation of hierarchically scheduled multiprocessor embedded systems**. *International Journal of Parallel Programming*, **36**(1):37–67, 2008. 19, 34

[68] Sivakumar Thangamuthu, Nicola Concer, Pieter JL Cuijpers, and Johan J Lukkien. **Analysis of ethernet-switch traffic shapers for in-vehicle networking applications**. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*, pages 55–60. IEEE, 2015. 19, 33

[69] Daniel Thiele, Rolf Ernst, and Jonas Diemer. **Formal worst-case timing analysis of Ethernet TSN's time-aware and peristaltic shapers**. In *Vehicular Networking Conference (VNC), 2015 IEEE*, pages 251–258. IEEE, 2015. 19, 33, 116

[70] Daniel Thiele and Rolf Ernst. **Formal worst-case performance analysis of time-sensitive ethernet with frame preemption**. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, pages 1–9. IEEE, 2016. 19, 33

[71] Daniel Thiele and Rolf Ernst. **Formal analysis based evaluation of software defined networking for time-sensitive ethernet**. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pages 31–36. IEEE, 2016. 20, 33

[72] Fedor Smirnov, Michael Glass, Felix Reimann, and Jürgen Teich. **Formal timing analysis of non-scheduled traffic in automotive scheduled TSN networks**. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1643–1646. IEEE, 2017. 20, 33

[73] Dorin Maxim and Ye-Qiong Song. **Delay Analysis of AVB traffic in Time-Sensitive Networks (TSN)**. In *RTNS 2017-International Conference on Real-Time Networks and Systems*, page 10, 2017. 20, 33

[74] Giuliana Alderisi, Gaetano Patti, and Lucia Lo Bello. **Introducing support for scheduled traffic over IEEE audio video bridging networks**. In *Emerging Technologies & Factory Automation (ETFA), 2013 IEEE 18th Conference on*, pages 1–9. IEEE, 2013. 20

[75] 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*, **53**(4):526–577, 2017. 20, 32

[76] Hyung-Taek Lim, Kay Weckemann, and Daniel Herrscher. **Performance Study of an In-Car Switched Ethernet Network without Prioritization.** In *Nets4Cars/Nets4Trains*, pages 165–175. Springer, 2011. 20, 34

[77] Jörn Migge, France RealTime-at Work, and Nicolas Navet. **Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks**. 2017. 20

[78] Hauke Stähle. *A Model-Based Framework for System-Wide Plug-and-Play with Flexible Timing Verification for Automotive Systems*. PhD thesis, Universitätsbibliothek der TU München, 2016. 21, 33

[79] Jonas Diemer, Jonas Rox, and Rolf Ernst. **Modeling of ethernet avb networks for worst-case timing analysis**. *IFAC Proceedings Volumes*, **45**(2):848–853, 2012. 21, 26, 34

[80] Takero Arai, Y Aiyama, Y Maeda, M Sugi, and J Ota. **Agile assembly system by plug and produce**. *CIRP Annals-Manufacturing Technology*, **49**(1):1–4, 2000. 21

[81] Tamio Arai, Yasumichi Aiyama, Masao Sugi, and Jun Ota. **Holonic assembly system with Plug and Produce**. *Computers in Industry*, **46**(3):289–299, 2001. 21

[82] François Jammes and Harm Smit. **Service-oriented paradigms in industrial automation**. *IEEE Transactions on Industrial Informatics*, **1**(1):62–70, 2005. 21, 34

[83] Paulo Pedreiras, Luis Almeida, and Paolo Gai. **The FTT-Ethernet protocol: Merging flexibility, timeliness and efficiency**. In *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, page 152. IEEE Computer Society, 2002. 21, 35

[84] Paulo Pedreiras, Paolo Gai, Luís Almeida, and Giorgio C Buttazzo. **FTT-Ethernet: A flexible real-time communication protocol that supports dynamic QoS management on Ethernet-based systems**. *IEEE Transactions on industrial informatics*, **1**(3):162–172, 2005. 21

[85] G Reinhart, S Krug, S Hüttner, Z Mari, F Riedelbauch, and M Schlögel. **Automatic configuration (plug & produce) of industrial ethernet networks**. In *Industry Applications (INDUSCON), 2010 9th IEEE/IAS International Conference on*, pages 1–6. IEEE, 2010. 22, 34

[86] Lars Dürkop, Henning Trsek, Jürgen Jasperneite, and Lukasz Wisniewski. **Towards autoconfiguration of industrial automation systems: A case study using Profinet IO**. In *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*, pages 1–8. IEEE, 2012. 22, 32, 34

[87] Lars Dürkop, Jahanzaib Imtiaz, Henning Trsek, Lukasz Wisniewski, and Jürgen Jasperneite. **Using opc-ua for the autoconfiguration of real-time ethernet systems**. In *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*, pages 248–253. IEEE, 2013. 22, 34

[88] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture*. Springer Science & Business Media, 2009. 22

[89] Nadine Keddis, Gerd Kainz, Christian Buckl, and Alois Knoll. **Towards adaptable manufacturing systems**. In *Industrial Technology (ICIT), 2013 IEEE International Conference on*, pages 1410–1415. IEEE, 2013. 22, 34

[90] Christian Buckl, Michael Geisinger, Dhiraj Gulati, Fran J Ruiz-Bertol, and Alois Knoll. **CHROMOSOME: A run-time environment for plug & play-capable embedded real-time systems**. *ACM SIGBED Review*, **11**(3):36–39, 2014. 22, 26, 29, 34

[91] Veit Hammerstingl and Gunther Reinhart. **Unified Plug&Produce architecture for automatic integration of field devices in industrial environments**. In *Industrial Technology (ICIT), 2015 IEEE International Conference on*, pages 1956–1963. IEEE, 2015. 22, 33

[92] Daniel Regulin, Amelia Glaese, Stefan Feldmann, Daniel Schütz, and Birgit Vogel-Heuser. **Enabling flexible automation system hardware: Dynamic reconfiguration of a real-time capable field-bus**. In *Industrial Informatics (INDIN), 2015 IEEE 13th International Conference on*, pages 1198–1205. IEEE, 2015. 22, 33

[93] Lars Durkop, Jurgen Jasperneite, and Alexander Fay. **An analysis of real-time ethernets with regard to their automatic configuration**. In *IEEE World Conference on Factory Communication Systems (WFCS*, pages 1–8. IEEE, 2015. 23, 33

[94] Peter Heise, Marc Lasch, Fabien Geyer, and Roman Obermaisser. **Self-configuring real-time communication network based on OpenFlow**. In *Local and Metropolitan Area Networks (LANMAN), 2016 IEEE International Symposium on*, pages 1–6. IEEE, 2016. 23, 33

[95] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. **Software-defined networking: A comprehensive survey**. *Proceedings of the IEEE*, **103**(1):14–76, 2015. 23

[96] Marina Gutiérrez, Wilfried Steiner, Radu Dobrin, and Sasikumar Punnekkat. **A configuration agent based on the time-triggered paradigm for real-time networks**. In *Factory Communication Systems (WFCS), 2015 IEEE World Conference on*, pages 1–4. IEEE, 2015. 23, 32, 33

[97] Wilfried Steiner, Pablo Gutiérrez Peón, Marina Gutiérrez, Ayhan Mehmed, Guillermo Rodriguez-Navas, Elena Lisova, and Francisco Pozo. **Next generation real-time networks based on it technologies**. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, pages 1–8. IEEE, 2016. 23, 32, 33

[98] **TwinCAT**. http://www.beckhoff.de/. Accessed: 02.01.2018. 23, 33

[99] **Automation Studio**. https://www.br-automation.com/. Accessed: 02.01.2018. 23, 33

[100] **Introducing Hirschmann Industrial HiVision**. http://www.hivision.de/. Accessed: 02.01.2018. 24, 33

[101] **Network Scheduler Slate XNS**. https://www.tttech.com/. Accessed: 02.01.2018. 24, 31, 32

[102] Franz Huber, Bernhard Schätz, Alexander Schmidt, and Katharina Spies. *AutoFocus — A tool for distributed systems specification*, pages 467–470. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. 25, 35

[103] Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F Gritzner, and Rainer Weber. *The design of distributed systems: an introduction to focus*. Citeseer, 1992. 25, 35

[104] Kim G. Larsen, Paul Pettersson, and Wang Yi. **Uppaal in a nutshell**. *International Journal on Software Tools for Technology Transfer*, **1**(1):134–152, 1997. 25, 29, 35

[105] Daniel Witsch, Birgit Vogel-Heuser, Jean-Marc Faure, and Gaëlle Marsal. **Performance analysis of industrial Ethernet networks by means of timed model-checking**. *IFAC Proceedings Volumes*, **39**(3):101–106, 2006. 25, 34

[106] Steve Limal, Stéphane Potier, Bruno Denis, and Jean-Jacques Lesage. **Formal verification of redundant media extension of Ethernet PowerLink**. In *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, pages 1045–1052. IEEE, 2007. 25, 34

[107] Jalil Boudjadar, Kim Guldstrand Larsen, Jin Hyun Kim, and Ulrik Nyman. **Compositional schedulability analysis of an avionics system using UPPAAL**. In *International Conference on Advanced Aspects of Software Engineering*, 2014. 25, 34

[108] Hassan Gomaa. **Designing Concurrent, Distributed, and Real-time Applications with UML**. In *IEEE Proc. of the 23rd International Conference on Software Engineering*, ICSE '01, pages 737–738, 2001. 25, 35

[109] James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004. 25

[110] Joseph T Buck, Soonhoi Ha, Edward A Lee, and David G Messerschmitt. **Ptolemy: A framework for simulating and prototyping heterogeneous systems**. 1994. 25, 27, 35

[111] **Stateflow**. https://de.mathworks.com/. (Accessed: 10.04.2017). 25, 33

[112] Tim Weilkiens. *Systems engineering with SysML/UML: modeling, analysis, design*. Morgan Kaufmann, 2011. 25, 33

[113] Ermeson Andrade, Paulo Maciel, Gustavo Callou, and Bruno Nogueira. **A methodology for mapping sysml activity diagram to time petri net for requirement validation of embedded real-time systems with energy constraints**. In *Digital Society, 2009. ICDS'09. Third International Conference on*, pages 266–271. IEEE, 2009. 26

[114] Ernesto Wandeler and Lothar Thiele. **Real-Time Calculus (RTC) Toolbox**. `http://www.mpa.ethz.ch/Rtctoolbox`. 26, 29, 33

[115] Henrik Schioler, Hans P Schwefel, and Martin B Hansen. **Cync: a matlab/simulink toolbox for network calculus**. In *Proceedings of the 2nd international conference on Performance evaluation methodologies and tools*, page 60. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2007. 26, 34

[116] Morteza Hashemi Farzaneh, Stefan Feldmann, Christoph Legat, Jens Folmer, and Birgit Vogel-Heuser. **Modeling Multicore Programmable Logic Controllers in Networked Automation Systems**. In *Industrial Electronics Society, IECON 2013-39th Annual Conference of the IEEE*, pages 4398–4403. IEEE, 2013. 26, 34

[117] Birgit Vogel-Heuser, Stefan Feldmann, Thomas Werner, and Christian Diedrich. **Modeling network architecture and time behavior of Distributed Control Systems in industrial plant automation**. In *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society*, pages 2232–2237. IEEE, 2011. 26, 34

[118] **Data-Centric Middleware - Real-Time Innovations**. https://www.rti.com/. Accessed: 22.09.2017. 27, 94

[119] **OpenDDS**. http://opendds.org/. Accessed: 22.09.2017. 27

[120] Linh TX Phan, Jaewoo Lee, Arvind Easwaran, Vinay Ramaswamy, Sanjian Chen, Insup Lee, and Oleg Sokolsky. **CARTS: a tool for compositional analysis of real-time systems**. *ACM SIGBED Review*, **8**(1):62–63, 2011. 27, 34

[121] Insik Shin and Insup Lee. **Periodic resource model for compositional real-time guarantees**. In *Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE*, pages 2–13. IEEE, 2003. 27, 35

[122] G. Gupta and E. Pontelli. **A constraint-based approach for specification and verification of real-time systems**. In *Proc. Real-Time Systems Symposium*, pages 230–239, 1997. 27, 35

[123] Neda Saeedloei and Gopal Gupta. **A Logic-based Modeling and Verification of CPS**. *SIGBED Rev.*, **8**(2):31–34, 2011. 27

[124] Neda Saeedloei and Gopal Gupta. **A Methodology for Modeling and Verification of Cyber-physical Systems Based on Logic Programming**. *SIGBED Rev.*, **13**(2):34–42, 2016. 27, 33

[125] Boon Thau Loo, Tyson Condie, Joseph M Hellerstein, Petros Maniatis, Timothy Roscoe, and Ion Stoica. **Implementing declarative overlays**. In *ACM SIGOPS Operating Systems Review*, **39**, pages 75–90. ACM, 2005. 27, 34

[126] Boon Thau Loo, Joseph M Hellerstein, Ion Stoica, and Raghu Ramakrishnan. **Declarative routing: extensible routing with declarative queries**. In *ACM SIGCOMM Computer Communication Review*, **35**, pages 289–300. ACM, 2005. 27, 34

[127] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. **Planetlab: an overlay testbed for broad-coverage services**. *ACM SIGCOMM Computer Communication Review*, **33**(3):3–12, 2003. 28

[128] Boon Thau Loo, Tyson Condie, Minos Garofalakis, David E. Gay, Joseph M. Hellerstein, Petros Maniatis, Raghu Ramakrishnan, Timothy Roscoe, and Ion Stoica. **Declarative Networking**. *Commun. ACM*, **52**(11):87–95, 2009. 28, 34

[129] Anduo Wang, Prithwish Basu, Boon Thau Loo, and Oleg Sokolsky. **Declarative network verification**. In *International Symposium on Practical Aspects of Declarative Languages*, pages 61–75. Springer, 2009. 28, 34

[130] Timothy L. Hinrichs, Natasha S. Gude, Martin Casado, John C. Mitchell, and Scott Shenker. **Practical Declarative Network Management**. In *Proc. of the 1st ACM Workshop on Research on Enterprise Networking*, WREN '09, New York, NY, USA, 2009. ACM. 28, 34

[131] Nuno P Lopes, Juan A Navarro, Andrey Rybalchenko, and Atul Singh. **Applying prolog to develop distributed systems**. *Theory and Practice of Logic Programming*, **10**(4-6):691–707, 2010. 28

[132] Morteza Hashemi Farzaneh, Sina Shafaei, and Alois Knoll. **Formally verifiable modeling of in-vehicle time-sensitive networks (TSN) based on logic programming**. In *Vehicular Networking Conference (VNC), 2016 IEEE*, pages 1–4. IEEE, 2016. 28, 29, 31, 33, 39, 45, 46, 50, 70

[133] Morteza Hashemi Farzaneh, Stefan Kugele, and Alois Knoll. **A graphical modeling tool supporting automated schedule synthesis for time-sensitive networking**. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Sept 2017. 29, 31, 32, 33, 38, 39, 41, 74, 75

[134] Antonio Guerrero, Víctor A Villagrá, Jorge E López De Vergara, and Julio Berrocal. **Ontology-based integration of management behaviour and information definitions using SWRL and OWL**. In *Ambient Networks*, pages 12–23. Springer, 2005. 29

[135] Sven Van der Meer, Brendan Jennings, Declan O'Sullivan, David Lewis, and Nazim Agoulmine. **Ontology based policy mobility for pervasive computing**. 2005. 29

[136] John Keeney, David Lewis, Declan O'Sullivan, Antoine Roelens, Vincent Wade, Aidan Boran, and Ray Richardson. **Runtime semantic interoperability for gathering ontology-based network context**. In *10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 56–65, 2006. 29

[137] Pradeep Ray, N Parameswaran, John Strassner, et al. **Ontology mapping for the interoperability problem in network management**. *IEEE Journal on Selected Areas in Communications*, **23**(10):2058–2068, 2005. 29

[138] Jorge E López De Vergara, Antonio Guerrero, Víctor A Villagrá, and Julio Berrocal. **Ontology-based network management: study cases and lessons learned**. *Journal of Network and Systems Management*, **17**(3):234–254, 2009. 29

[139] Arosha K Bandara, Antonis Kakas, Emil C Lupu, and Alessandra Russo. **Using argumentation logic for firewall policy specification and analysis**. In *Large Scale Management of Distributed Systems*, pages 185–196. Springer, 2006. 29

[140] Torsten Klie, Felix Gebhard, and Stefan Fischer. **Towards automatic composition of network management web services**. In *10th IFIP/IEEE International Symposium on Integrated Network Management*, pages 769–772, 2007. 29

[141] A Martinez, M Yannuzzi, JE Lopez de Vergara, R Serral-Gracia, and W Ramirez. **An Ontology-Based Information Extraction System for bridging the configuration gap in hybrid SDN environments**. In *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 441–449, 2015. 29, 33

[142] Morteza Hashemi Farzaneh and Alois Knoll. **An ontology-based plug-and-play approach for in-vehicle Time-Sensitive Networking (TSN)**. In *Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual*, pages 1–8. IEEE, 2016. 29, 32, 33, 76, 93

[143] M. GutiÃ©rrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat. **Self-configuration of IEEE 802.1 TSN networks**. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, Sept 2017. 32

[144] **Symtavision**. https://auto.luxoft.com/. Accessed: 22.01.2018. 33

[145] Giuliana Alderisi, Giancarlo Iannizzotto, and Lucia Lo Bello. **Towards IEEE 802.1 Ethernet AVB for Advanced Driver Assistance Systems: a preliminary assessment**. *Proc. of 2012 IEEE 17th International Conference on Emerging Technologies & Factory Automation (ETFA 2012)*, page 4, 2012. 45, 46

[146] Martin Buechel, Jelena Frtunikj, Klaus Becker, Stephan Sommer, Christian Buckl, Michael Armbruster, Andre Marek, Andreas Zirkler, Cornel Klein, and Alois Knoll. **An automated electric vehicle prototype showing new trends in automotive architectures**. In *18th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1274–1279. IEEE, 2015. 50

[147] William F. Dowling and Jean H. Gallier. **Linear-time algorithms for testing the satisfiability of propositional horn formulae**. *The Journal of Logic Programming*, **1**(3):267 – 284, 1984. 50

[148] David L Poole and Alan K Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010. 51

[149] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Higher Ed, 2016. 51

[150] Clark W Barrett, Roberto Sebastiani, Sanjit A Seshia, and Cesare Tinelli. **Satisfiability Modulo Theories.** *Handbook of satisfiability*, **185**:825–885, 2009. 52

[151] Bruno Dutertre and Leonardo De Moura. **The yices smt solver**. *Tool paper at http://yices. csl. sri. com/tool-paper. pdf*, **2**(2):1–2, 2006. 52

[152] Leonardo De Moura and Nikolaj Bjørner. **Z3: An efficient SMT solver**. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008. 52

[153] Rob Enns. **NETCONF configuration protocol**. 2006. 77

[154] William Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc., 1998. 77

[155] Andre Suelflow, Goerschwin Fey, Roderick Bloem, and Rolf Drechsler. **Using unsatisfiable cores to debug multiple design errors**. In *Proceedings of the 18th ACM Great Lakes symposium on VLSI*, pages 77–82. ACM, 2008. 80

[156] Mark H Liffiton and Ammar Malik. **Enumerating infeasibility: Finding multiple MUSes quickly**. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 160–175. Springer, 2013. 80, 81, 83

[157] Mark Liffiton. **MARCO**. https://sun.iwu.edu/ mliffito/marco/. Accessed: 25.08.2018. 81

[158] Ofer Guthmann, Ofer Strichman, and Anna Trostanetski. **Minimal unsatisfiable core extraction for SMT**. In *Formal Methods in Computer-Aided Design (FMCAD), 2016*, pages 57–64. IEEE, 2016. 81, 83

[159] John W Chinneck and Erik W Dravnieks. **Locating minimal infeasible constraint sets in linear programs**. *ORSA Journal on Computing*, **3**(2):157–168, 1991. 81

[160] **HSmtMuc**. https://github.com/HSmtMuc/HSmtMuc. Accessed: 25.08.2018. 81

[161] Narendra Jussien and Samir Ouis. **User-friendly explanations for constraint programming**. *arXiv preprint cs/0111037*, 2001. 81

[162] Francisco Pozo. *Synthesis of Extremely Large Time-Triggered Network Schedules*. PhD thesis, Mälardalen University, 2017. 88

[163] Stefan Kehrer. **What Does TSN Configuration Look Like Today and In the Future?** https://www.belden.com/. Accessed: 08.09.2018. 93

[164] Gerardo Pardo-Castellote. **Omg data-distribution service: Architectural overview**. In *Proc. 23rd International Conference on Distributed Computing Systems Workshops*, pages 200–206. IEEE, 2003.

[165] Iwan Schafer and Max Felser. **Topology discovery in PROFINET**. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 704–707. IEEE, 2007. 98

[166] Tatu Ylonen and Chris Lonvick. **The secure shell (SSH) protocol architecture**. 2006. 103

[167] Morteza Hashemi Farzaneh and Alois Knoll. **Time-sensitive networking (TSN): An experimental setup**. In *2017 IEEE Vehicular Networking Conference (VNC)*, pages 23–26, Nov 2017. 105

[168] Rosebrock Adrian. **Facial landmarks and drowsiness detection with OpenCV and dlib.** https://www.pyimagesearch.com. Accessed: 08.09.2018. 107

[169] **Ostinato Network Traffic Generator**. https://ostinato.org/. Accessed: 22.01.2018. 108