Design Optimization of IEEE Time-Sensitive Networks (TSN) for Safety-Critical and Real-Time Applications

Gavrilut, Voica Maria

Publication date:
2018

Document Version
Publisher's PDF, also known as Version of record

Citation (APA):
Gavrilut, V. M. (2018). *Design Optimization of IEEE Time-Sensitive Networks (TSN) for Safety-Critical and Real-Time Applications*. DTU Compute. DTU Compute PHD-2018 Vol. 500

Design Optimization of IEEE Time-Sensitive Networks (TSN) for Safety-Critical and Real-Time Applications

Voica Gavriluț

Kongens Lyngby 2018
PhD-2018-500

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk
PhD-2018-500
ISSN: 0909-3192

Summary (English)

A safety-critical cyber-physical system (CPS) is a system that will not endanger human life or the environment, or is intended to prevent such harm. Many safety-critical systems are also real-time, where the correctness depends, in addition to the validity of results, on the time instance at which they are produced. This thesis addresses safety-related distributed CPSes, interconnected using the communication protocol colloquially known as Time-Sensitive Networking (TSN).

Ethernet, although is low cost and has high speeds, is known to be unsuitable for real-time and safety-critical applications. Therefore, standards such as TSN have been proposed to extend switched Ethernet in order to guarantee reliable and time-predictable communication. In a TSN-based network, the interacting nodes, known also as End Systems (ESes), are interconnected by full-duplex physical links and network switches. The data in TSN is exchanged via streams.

TSN is highly suitable for applications of different safety-criticality levels (highly critical, mission critical, non-critical), as it offers several traffic types, such as Time-Triggered (TT) and Audio-Video Bridging (AVB) traffic types. TT has the highest priority and is sent based on schedule tables, called Gate Control Lists (GCLs). By synthesizing carefully the GCLs, TT messages can have low end-to-end latency and low jitter. AVB is an asynchronous traffic type that is intended for applications that require bounded end-to-end latencies, but has a lower priority than TT traffic.

Regarding dependability, we assume that the engineer will specify for each application, depending on its criticality, the required redundancy level. This translates, at network topology level, into requirements for redundant disjoint routes between the devices involved in the communication. In this context, we focus on *synthesizing a low-cost fault-tolerant network architecture*, which can guarantee the safety and real-time requirements of the applications. We also solve the problem of routing disjoint redundant streams on the synthesized architecture.

Similar to the debate in real-time systems between time-triggered and event-triggered implementations there is no agreement on the appropriate traffic type for the messages of mixed-criticality applications (e.g., TT or AVB). Hence, we have also addressed the problem of *traffic type assignment* for mixed-criticality messages in TSN. We decide, for each message, if it should use the TT or AVB traffic type, such that the hard real-time messages meet their deadlines and soft real-time messages maximize their quality-of-service.

Although researchers have started to propose approaches for the routing and scheduling (i.e., GCL synthesis) of TT traffic, all previous research has ignored lower priority real-time traffic such as AVB, resulting in TT configurations that may increase the worst-case delays of AVB traffic, rendering it unschedulable. Hence, we have also proposed a *joint routing and scheduling approach for TT traffic*, which takes into account the AVB traffic, such that both TT and the AVB traffic are schedulable.

The work in this thesis has been implemented as software tools, which have been extensively evaluated on a large number of synthetic as well as realistic test cases.

Summary (Danish)

Et sikkerhedskritisk cyber-fysisk system (CPS) er et system, der er designet så det ikke kan bringe menneskeliv eller miljø i fare. Mange sikkerhedskritiske systemer er også realtidssystemer, idet korrektheden afhænger af resultaternes gyldighed på det tidspunkt, hvor de produceres. Denne afhandling omhandler sikkerhedskritiske fysisk distribuerede cyber-fysiske systemer, hvor de computere der benyttes er forbundet vha. en variant af Ethernet der tilbyder såkaldt time sensitive networking (TSN).

Ethernet er kendetegnet ved lav pris og høj båndbredde, og det er derfor meget udbredt. Desværre er det helt uegnet til brug i sikkerhedskritiske applikationer. Der er derfor foreslået forskellige overbygninger, der kan sikre og garantere pålidelig og tidsforudsigelig kommunikation. TSN er et eksempel. Det er standardiseret af IEEE og får stigende udbredelse. I et TSN-baseret netværk er de kommunikerende enheder, kaldet end systems, forbundet med hinanden ved hjælp af fuld duplex links og netværk-switches. Data udveksles i TSN netværk via streams.

TSN er velegnet til applikationer med flere forskellige sikkerhedsniveauer (yderst kritisk, missionskritisk, ikke-kritisk) idet det understøtter forskellige trafikklasser, bl.a. time-triggered (TT) trafik og Audio-Video Bridging (AVB) trafik. TT trafik har højest prioritet og bliver sendt på faste tidspunkter efter en slags køreplaner – såkaldte Gate Control Lists (GCLs). Disse kan optimeres så TT-trafik oplever lav latenstid og lav jitter. AVB er en asynkron trafiktype, der er beregnet til applikationer, der kræver begrænsede end-to-end latenstid, men har lavere prioritet end TT-trafik.

Mht. pålidelighed antager vi, at der for hver applikation forelægger en specifikation af kritikalitet. Herfra kan der så afledes specifikationer for disjunkte og redundante signalveje mellem de kommunikerende enheder i systemet. I denne sammenhæng fokuserer afhandlingen på at syntetisere prisbillige og fejltolerante netværksarkitekturer, som kan garantere opfyldelse af de sikkerheds- og realtidskrav der stilles.

Inden for fagområdet realtidssystemer er der en stående og uafklaret diskussion af to alternative paradigmer for systemernes grundlæggende virkemåde: time-triggered eller event-triggered. En tilsvarende uafklarethed gør sig gældende mht. hvordan data hørende til forskellige trafikklasser (f.eks. TT eller AVB) skal håndteres. Afhandlingen tager også fat i dette emne, og undersøger hvordan TSN-netværk kan håndtere data hørende til forskellige trafikklasser, her specifikt TT og AVB, således at realtids trafik leveres inden for de krævede deadlines og således at kvaliteten ad den øvrige trafik maksimeres.

Selv om forskere er begyndt at foreslå fremgangsmåder til routing og planlægning (dvs. GCL-syntese) af TT-trafik, har alle tidligere undersøgelser ignoreret lavere prioriteret realtidstrafik som AVB, hvilket resulterer i TT-konfigurationer, der kan øge de værste forsinkelser af AVB-trafik, og i nogle tilfælde gøre det umuligt at finde (tilfredsstillende) løsninger. Som en løsning på dette forhold foreslår afhandlingen en samlet løsning der tilgodeser og optimerer både TT og AVB trafik.

Metoderne udviklet i denne afhandling er blevet implementeret som softwareværktøjer og disse er evalueret på at stort antal syntetiske og virkelige testeksempler.

Preface

This thesis was prepared at DTU Compute in fulfilment of the requirements for acquiring an Ph.D. degree in computer engineering.

In this thesis, we propose methods and tools to support automatic design space exploration for the design of computer networks that have to accommodate safety-critical real-time applications. The thesis consists of an introductory chapter and three papers.

The work has been supervised by Professor Paul Pop and co-supervised by Associate Professor Christian Damsgaard Jensen.

Lyngby, 31-October-2018

Voica Gavriluţ

[illegible]

Acknowledgements

I would like to express my sincere respect and deepest gratitude to my main supervisor, Prof. Paul Pop, for the following three most important things. Firstly, I highly appreciate him for accepting me as his PhD student, despite my disability, giving me thus the opportunity to develop my skills and knowledge. I am also thankful for his constant and constructive feedback. And lastly, I cannot thank him enough for restoring my hope by showing me how to be encouraging, patient and kind with people, doing meanwhile his job with professionalism.

I would like to express also a big thank you to my co-authors, and specially to Adj. Prof. Soheil Samii for his industry-oriented viewpoint. Special appreciation goes also towards Bahram Zarrin and Luxi Zhao, who were also my office mates, for our thought provoking, fruitful and supportive discussions.

I would like to thank Prof. Hans Hansson and Assoc. Prof. Elisabeth Uhlemann, from Mälardalen University (MDH), Sweden, for their hospitality and all their guidance during my external stay. I enjoyed so much the friendly atmosphere and various and insightful conversations with the people from MDH. I am also thankful for the ideas and different viewpoints and for the openings for collaboration that they provided.

I highly appreciate the support provided by the administrative and technical staff. Special acknowledgments are expressed to Prof. Jens Sparsø for the Danish translation and Karin Tunder and Ellen Juel Nielsen, the people behind the scene who were always available to help me to navigate through the bureaucratic system.

I would also like to thank all my friends and all the fellow students for their support and help in different aspects through these three years. I would like to express my warm thanks to Adriana Zsurzsan, Olivia Perederic and Paula Muntianu for listening me patiently my complains when I was discouraged, and for being always there to celebrate each achievement. Also very helpful were the discussions with Domiţian Tămaş-Selicean, about the PhD life and life in Denmark in general. Discreet but valuable presences were also my friends Patricia and Erwin Karalyos who were constantly cultivating my passion for IQ and board games.

Finally, I take this opportunity to address my heartfelt thanks to my family for their endless support. Special gratitude I have for my parents, Valeriu and Doina Gavriluţ, who supported me to take this way, and who, with unlimited love, encouraged me when I felt hopeless.

Contents

List of Algorithms

List of Figures

List of Tables

Abbreviations

Abbreviation	Meaning
ADAS	Advanced Driver Assistance System
AFDX	Avionics Full-DupleX switched-Ethernet
ALAP	As-Late-As-Possible
ARL	Address Resolution Logic
ASAP	As-Soon-As-Possible
ATS	Asynchronous Traffic Shaping
AVB	Audio-Video Bridging
BAG	Bandwidth Allocation Gap
BE	Best-Effort
BFS	Breadth-First Search
Bs	Bridges
CAD	Computer-Aided Design
CBS	Credit-Based Shaper
CEV	Crew Exploration Vehicle
COTS	Commercial Off-The-Shelf
CP	Constraint Programming
CPS	Cyber-Physical System
CRC	Cyclic Redundancy Check
DE	Deterministic Ethernet
DEI	Drop Eligible Indicator
DL	Dataflow Link
DP	Dataflow Path
ECU	Electronic Control Units
ES	End System
ET	Event-Triggered
ETG	EtherCAT Technology Group

Abbreviation	Meaning
GCL	Gate Control List
GRASP	Greedy Randomized Adaptive Search Procedure
HRT	Hard Real-Time
IEEE	Institute of Electrical and Electronics Engineers
ILP	Integer Linear Programming
ITS	Intelligent Transport System
JRS	Joint Routing and Scheduling
KSP	K-Shortest Path
LAN	Local Area Network
LIDAR	LIght Detection And Ranging
LNS	Large Neighborhood Search
MAC	Medium Access Control
MBSE	Model-Based Systems Engineering
MC	Modify Class
MGCL	Modify GCL
MII	Media-Independent Interface
MIAT	Minimum Inter-Arrival Time
MTU	Maximum Transmission Unit
NC	Non-Critical
NRE	Non-Recurring Engineering cost
NS	Network Switch
OMT	Optimization Modulo Theory
PAR	Project Authorization Request
PC	Personal Computer
PCP	Priority Code Point
PE	Processing Element
PHY	PHYsical connector
QoS	Quality-of-Service
RC	Rate-Constrained
RCL	Restricted Candidate List
RL	Redundancy Level
RSTP	Rapid Spanning Tree Protocol
SMT	Satisfiability Modulo Theory
SPB	Shortest Path Bridging
SRP	Stream Reservation Protocol
SRT	Soft Real-Time
STT	Switch Traffic Type
TAS	Time-Aware Shaper
TCI	Tag Control Information
TDM	Time-Division Multiplexing
TPID	Tag Protocol IDentifier
TRH	Topology and Routing Heuristic
TRO	Topology and Routing Optimization

Abbreviation	Meaning
TS	Tabu Search
TSA	Transmission Selection algorithm
TSN	Time-Sensitive Networking
TT	Time-Triggered
TTA	Traffic Type Assignment
TTP	Time-Triggered Protocol
UBS	Urgency-Based Scheduler
VLAN	Virtual Local Area Network
VLID	VLAN IDentifier
WCD	Worst-Case Delay
WCPHD	Worst-Case Per-Hop Delay

CHAPTER 1

Introduction

1.1 Motivation and Background

Computer systems are everywhere in our life. In addition to the general-purpose computers, such as Personal Computers (PCs), laptops, and servers, there are many "invisible computers" that are *embedded* into devices and systems that we use. For example, today's high-end cars may use more than 100 microcontrollers, which control most aspects of a car's functionality. These *embedded computer systems* are interconnected with each other and to the internet. They are used in different application areas: from intelligent entertainment systems to power and nuclear plants, from smart toys to medical devices, from office suppliers to automobiles and aircraft. This thesis is concerned with *real-time* applications implemented on *safety-critical cyber-physical systems.*

Cyber-physical systems. Embedded computer systems typically take inputs from the environment (via sensors) and act upon it (via actuators), as is the case with a temperature or pressure controller in a steel furnace for example. To denote such systems, Helen Gill, at the National Science Foundation in the U.S.A., introduced the term *Cyber-Physical System* (CPS) [ALAS15]. Hermann Kopetz describes in [Kop11] a CPS having three main components: (1) the controlled cluster (the physical components that are controlled), (2) the control cluster (the set of computers controlling the physical entities) and (3) the operator cluster (people controlling or supervising the system). CPSes are pervasive in our society, used in most areas, from automotive and aerospace systems to medical devices and industrial systems such as factories. CPSes are underpinning our "smart society", being used in Smart Cities, Intelligent Transport Systems (ITSes), Smart Manufacturing, Smart Grid, etc. Many of the societal challenges that we face, related to the environment and energy consumption, are currently tackled using cyber-physical systems as a key enabler technology.

Safety-critical systems. When a system failure endangers human life or the environment we say about such system that is safety-critical. For example in a steel furnace, a faulty pressure controller could lead to the furnace explosion, endangering the people operating the furnace. Knight observes that a safety-critical system is "when we depend on it for our well being" [Kni02]. Each application area may have its own specific definition; for example, the United States Department of Transportation, Aircraft Certification Division, defines a safety-critical system as a system that is losing less than 1 life in a billion (10^9) hours of operation [Uni88].

Real-time systems. CPSes are typically also real-time. There are several definitions of a real-time system, including the common misconception that a "system operates in real time if it is able to quickly react to external events" [But11]. Instead, a correct and broadly accepted definition formulated by Kopetz is that: "A real-time computer system is a computer system where the correctness of the system behavior depends not only on the logical results of the computations but also on the physical time when these results are produced" [Kop11]. In other words for a real-time system, the results must be produced within the deadlines imposed by computing system characteristics or the controlled environment [But11]. We can distinguish two types of deadlines: *hard* deadlines—where a result produced after the deadline is equivalent with a failure—and soft deadlines—where a result obtained after the deadline just degrades the quality [Kop11, BLAC05]. Furthermore, Kopetz defines as a *hard real-time system* or a safety-critical system any real-time system that has to meet at least one hard deadline. Throughout this thesis, we refer as hard or soft real-time messages to those messages having hard or soft deadlines.

Distributed architectures and TSN. CPSes are typically *distributed*, i.e., they consist of several computational nodes (each with its own computation entities, such as a CPU and cores, and memory) which communicate with each other in a network. The architecture of CPSes, implementing multiple (potentially real-time and safety-critical) applications, has evolved over time. In the beginning, *federated architectures* were used, where the applications were implemented by physically separated computation nodes. Currently, the design of CPSes use *integrated architectures*, where multicore computation units serve multiple control applications. Such an evolution can be observed in the automotive industry, where nowadays we have multi-purpose Electronic Control Units (ECUs) interconnected via backbone networks [LSS+12]. As presented in subsection 1.1.3, in this thesis we consider that the nodes are interconnected using IEEE 802.1 Time-Sensitive Networks (TSN) [TSN12], which is an emerging set of IEEE standards that extend the well-known Ethernet standard, with features for real-time and safety-critical applications.

1.1.1 System and Application Models

1.1.1.1 System Model and Example

Figure 1.1 shows a simplified representation of a distributed architecture of an automotive electronics system, which implements control applications that involve sensors, computation units and actuators [Kop11].

Throughout this thesis, the logical topology of a network is mathematically represented as an undirected graph with End Systems (ESes) and Network Switches (NSes) as nodes and communication connections as links. In general, an ES embodies a possible multi-core CPU, a memory and I/Os. It is broadly accepted that only the computation units carry out intelligence, but nowadays there are also smart sensors, such as cameras and radars, which can preprocess the sensed signals. As already stated for this example the ESes are represented by sensors, actuators and Electronic Control Units (ECUs). An NS, called in TSN bridge, has a different number of physical ports. Each such physical port is further composed of two logical ports; an ingress and an egress port, respectively. The term *network device* generally denotes either an end system or a network switch. In Figure 1.1 we show part of an intra-vehicle network with 13 ESes (5 sensors, 5 actuators and 3 ECUs) and 3 NSes (in this example, the NSes are built in the ECUs, which is typical for the automotive area). The connections among nodes are represented by physical full-duplex links.

Logically a physical link consists of two directed connections. Such a directed connection is further called Dataflow Link (DL), as $dl_1 = (sensor_1, ECU_1)$ in Figure 1.1. A group of such successively connected DLs, where the sequence starts and ends with an ES, represents a Dataflow Path (DP). In TSN the communication can be also multicast, therefore a message can be forwarded through a dataflow path (unicast case) or through a multicast tree — a set of multiple dataflow paths having the same source ES. Such a forwarding structure is called route. For example in Figure 1.1, two possible routes are $r_1 = [sensor_1, NS_1, [ECU_1, [NS_2, ECU_2]]]$ (a multicast tree) and $r_2 = [ECU_1, NS_1, actuator_1]$.

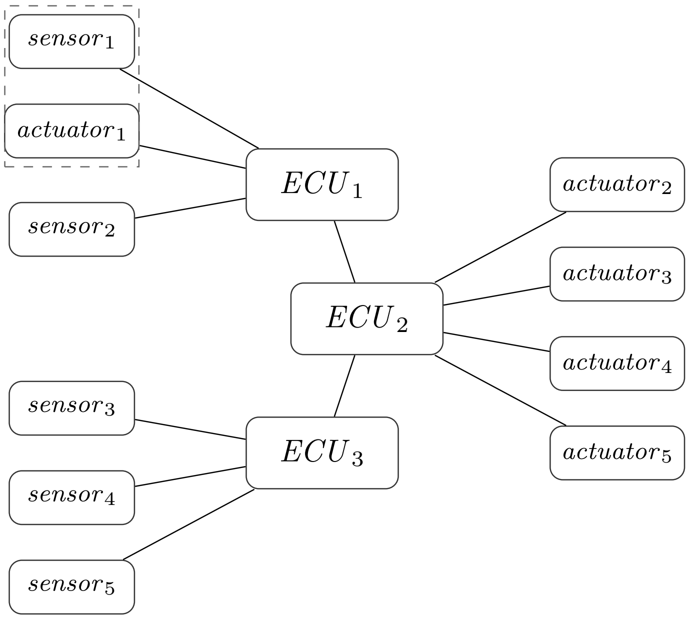


Figure 1.1: Part of an intra-vehicle network that consists on sensors, actuators and Electronic Control Units (ECUs)

1.1.1.2 Application Model and Example

Through this thesis, the *application model* captures the messages exchanged among applications. Such a model for our example is depicted in Table 1.1. Safety-critical messages denote the messages produced or consumed by at least one safety-critical application. Real-time messages are defined similarly, involving real-time applications.

In our model, a message has the following properties: source ES, destination(s) ESes, size, period and deadline. The meaning of source and destination end systems was already presented in subsubsection 1.1.1.1. The size represents the maximum number of bytes that a message can carry on. TSN is an Ethernet extension, therefore messages are packed and transmitted in Ethernet frames. Thus, if the size of a message exceeds the Ethernet-dictated Maximum Transmission Unit (MTU) a message is split into multiple frames.

Often real-time messages are transmitted repeatedly. This repetition is either periodic (or strictly periodic) — any successive message instances are separated by a fixed amount of time (or period) — or sporadic — between any successive message instances there is at least a minimum amount of time, called Minimum Inter-Arrival Time (MIAT). In our example from Table 1.1, we consider that all messages are periodic. The meaning of the deadline of a message has been previously presented in the context of real-time systems. Usually in the literature are considered the default deadlines for messages, i.e., the deadline is the same with the period, as in Table 1.1. But throughout this thesis, we consider the generic case with specific-application deadlines that can be different from the period.

Message	Source	Destination(s)	Size (in Bytes)	Period (in ms)
m_1	$sensor_1$	ECU_1 ECU_2	1200	2
m_2	ECU_1	$actuator_1$	100	1
m_3	$sensor_2$	ECU_1 ECU_2	1400	4
m_4	ECU_1	ECU_2	900	1
m_5	ECU_2	ECU_1	75	1
m_6	ECU_2	$actuator_2$ $actuator_3$ $actuator_4$ $actuator_5$	150	1

Table 1.1: A selected set of messages from an automotive application

Let us assume that the architecture in Figure 1.1 implements a safety-critical obstacle detection application, which uses the $sensor_1$, $actuator_1$ (components of a camera) and ECU_1 (the associated computation unit). $sensor_1$ is the camera sensor that sends raw data m_1 to ECU_1. The job of ECU_1 is to: (1) filter and process the raw data, (2) send the graphical information, as m_4, to other ECUs and (3) based on quality of the received raw images m_1 and vehicle state m_5 decide and send to $actuator_1$ the updated values m_2 to control the sensor exposure, as shutter speed or lens aperture.

1.1.1.3 Fault Model and Example

There are many types of faults that can affect the transmission of messages in distributed systems. Such transmission faults, encountered in the literature also as communication or network faults, can be accidental or intentionally produced and they can be caused by faults of the physical components or the software faults. This thesis focuses on the network design for safety-critical applications, assuming that the network is secure, i.e., the network engineers are already using the state-of-the-art security methods to prevent the intentional faults.

Accidental *software faults*, known also as bugs, are usually detected and corrected during the software development phase. A proper software development process involves, in addition to the actual implementation, intensive testing and the so-called bug fixing.

The most common type of faults are *physical faults*. The principal causes of this kind of faults are aging of the component or external factors as radiations, power transients, etc. [ALRL04]. For example, the vibration and thermal fluctuations cause the corrosion of the cable terminals. This corrosion leads to the most common faults in intra-vehicle networks, namely permanent faults in the physical links and their connectors [Sie04, Tib13].

Furthermore, the faults can be classified regarding their duration, therefore we are talking about *permanent* and *transient* faults [ML09]. To tolerate these types of faults there have been proposed several methods, such as redundancy and error-correction codes, respectively [ML09]. Furthermore, the redundancy can be either spatial—when a message is transmitted through redundant routes (routes that do not share any physical component)—or temporal (used to tolerate transient faults of physical components)—represents the retransmission of a message within its period. The reception of an erroneous message can be detected with the very known polynomial division based *Cyclic Redundancy Check (CRC)* schema. Before transmission to a message is attached a checksum, computed based on polynomial division. Upon reception, the checking value is computed again. If the newly computed checking value and the checksum are not the same, the receiver can request the retransmission of the message.

The camera-based application presented in Figure 1.1 implements a spatial fault-tolerant mechanism if the decision routine implemented in ECU_2 takes into account both, the processed and raw graphical data. In the fault-tolerant scenario, if the obstacle detection job implemented at ECU_1 is faulty the raw data provided through m_1 can be still used to gather some information about the environment.

1.1.2 Design Constraints

In order to function correctly, a cyber-physical system not only has to be designed such that it implements the required functionality, but also has to fulfill a wide range of competing design constraints: size, weight and power consumption, collectively referred to as SWaP; performance (latency, throughput); predictability; dependability, which integrates attributes such as reliability (continuity of correct service), availability (readiness for correct service), safety (absence of catastrophic consequences on the users and the environment), integrity (absence of improper system state alterations), confidentiality (absence of unauthorized disclosure of information) and maintainability (ability to undergo repairs and modifications); cost, which can be broken down into unit cost, the cost to produce one copy of the product, and Non-Recurring Engineering cost (NRE), the one-time engineering cost to design and develop the system; time-to-market, i.e., the necessary time to design and produce the system to the point it can be sold.

In this thesis, we are interested in the design metrics related to *timeliness* and *dependability.* To explain metrics, we will use the system example composed of the network depicted in Figure 1.1 and the set of messages from Table 1.1, that represents, as mentioned, a part of an automotive architecture.

1.1.2.1 Timeliness

Real-time safety-critical applications have strict timing requirements referred to as *deadlines.* In addition to deadlines, there are also other constraints that should be met by a highly critical system such as: *timeliness*, *dependability* and *fault-tolerance.*

Multiple timing metrics, such as delay and jitter [Kop11, But11], have to be considered when designing a safety-critical system.

Worst-Case Delay. The delay, or response time in context of hardware design [But11], is the time elapsed between ending and starting of an action. In the context of this thesis, we use the term *delay* to denote the message delay or transmission time. For example, the delay of the message m_3 is computed as the time when m_3 started its transmission from its source $sensor_2$ subtracted from the time when m_3 is received at its destination ECU_1. Factors such as switching delay or number of messages that are competing on the same egress port of a switch can increase the delay of a message. As highlighted by Buttazzo and Kopetz, delays in safety-critical systems have to be highly predictable. Is not enough to consider only the average delay, but the highly critical messages must meet their deadlines also in the worst-case scenarios [Kop11, But11]. Therefore, a safety-critical system must be designed such that even the maximum delay of a message, known also as *Worst-Case Delay* (WCD), should be smaller or equal to the message deadline. Furthermore, we say that a message is *schedulable* when it can meet its deadline even in the worst-case scenario; when the WCD is smaller than the deadline.

Jitter. The jitter of a message represents the difference between the WCD and the minimum delay of a message [Kop11]. Buttazzo considers that a system is predictable when the number of factors that impact the delay variations is minimized, i.e., when the jitter is low [But11]. In other words, a system is predictable when the WCDs can be computed and overall jitter is low. To illustrate the impact of the jitter and delay let us consider the camera example in Figure 1.1. As we can observe the m_1, ECU_1 job and m_2 creates a closed loop. In this context is obvious that a delay greater than the sampling period, the time between two shootings, leads to an under/overexposure of the sensor.

1.1.2.2 Dependability

In the Cambridge English dictionary, *dependability* is defined as "the quality of being able to be trusted and being very likely to do what people expect". In [Kop11], Kopetz defines dependability as "the quality of service a system delivers to its users during an extended period of time". In the following are defined four dependability attributes that are most relevant for the design of safety-critical systems.

Reliability. denoted $R(t)$ is the probability that the system is operational during the time interval t [Kop11, ALRL04].

Availability denoted $A(t)$ represents the fraction of the time interval t when the system is ready to deliver the correct service [Kop11, ALRL04].

Safety is a property of a system that will not endanger human life or the environment [ALRL04]. Furthermore, Kopetz introduces the concepts of safety modes: the malign and benign modes [Kop11]. If in our camera example, from Figure 1.1, the reliability of camera acquisition is too low, i.e., too many lost video frames and bad quality of images, the failure of the obstacle detection algorithm implemented in ECU_2, for example, could lead to a crash with the non-detected obstacle.

Fault Tolerance. Many safety-critical systems must be designed to be fault-tolerant, i.e., such that they deliver the correct service even in the case of faults [ALRL04].

1.1.3 Communication Protocols

1.1.3.1 Evolution of Communication Protocols

Several communication protocols were developed since 80s to meet the timeliness and dependability requirements of real-time safety-critical applications. In this section, most relevant protocols are compared in terms of network architecture and supported type of traffic. During the past 60 years or so, the machine-to-machine communication infrastructure has drastically evolved. Until beginning of years 2000s, the predominant protocols were based on bus architectures, as CAN [Rob86], Fieldbus [Int98], SAFEbus [Hon92], TTP [KG93], TTCAN [FMHH01], FTT-Ethernet [PGAB05] or FlexRay [Con06]. However, from years 2000s onwards, the "switched architecture" protocols have gained ground, such as ProfiNet [Int01], EtherCAT [Eth03], AFDX [Aer09], AVB [AVB05], Switched FlexRay [LCM11], TTEthernet [SAE11], TSN [TSN12] or AVB-ST [APLB13].

A notable advancement is from the 80s, when the computer interaction moved from point-to-point connections, where any two communicating machines should be connected by a cable, to bus-based architectures, where all machines from a network are attached to the same cable, known also as *communication bus* [TW11]. Even if the cabling and installation costs were reduced, in a bus architecture, comparing to the point-to-point architecture, this came at cost of decreased bandwidth. The topologies used for bus architectures are the bus/line topology, the ring topology and the star and snowflake/tree topologies, where multiple computers can be connected to a hub [TW11]. In a bus-based architecture, all communicating devices are competing for the same communication medium, saying that all devices are on the same collision domain. Therefore, new Medium Access Control (MAC) protocols were needed.

The next notable achievement, as documented by Tanenbaum [TW11], is from the mid 90s, when switched architectures were introduced, which can be seen as a trade-off between point-to-point and bus-based architectures in terms of cabling and bandwidth. Multiple computers are thus connected to a switch, but compared to a hub, now there are separate collision domains between the switch and each connected machine. The switched architectures are even more flexible in regard to supported topologies, therefore we can have star, snowflake and tree, bi- or n-torus, mesh or hybrid topologies.

1.1.3.2 Time-Triggered vs. Event-Triggered Solutions

The debate from real-time systems community according to Event-Triggered (ET) and Time-Triggered (TT) approaches [Kop11] is reflected also in the evolution of dependable communication protocols. The initial approach was to release the communication based on internal or external events. Therefore, in the beginning, regardless of the application area, from aerospace to industrial automation, the communication protocols, such as CAN, Fieldbus or SAFEbus, could handle only the ET traffic. Already at the beginning of 90s, namely in 1991, Hermann Kopetz proposed a comparative analysis of ET and TT approaches in terms of predictability, resource allocation and system scalability[Kop91]. This analysis is followed in 1993 by [KG93], which introduced the Time-Triggered Protocol (TTP).

As shown in [Kop91] and [Kop11] TT-based architectures are more predictable, but ET-based architectures have more reduced installation costs and allocated resources. As communication systems start to grow in size and complexity, after the 2000s, “mixed” architectures started to appear, where both type of traffic, ET and TT, are present. Besides TTP and EtherCAT which are purely TT, the majority of protocols after the year 2000 (ProfiNet, TTCAN, FTT-Ethernet, FlexRay, TTEthernet, TSN and AVB-ST) being mixed. After the emergence of TT paradigm, only a few dependable protocols are exclusive ET, such as AFDX and AVB.

1.1.3.3 IEEE 802.1 Time-Sensitive Networking (TSN)

Although in different areas, such as automotive, airspace or industrial automation, there are specialized safety-critical protocols, the trend is towards standardization. In the beginning, each area developed specific protocols, therefore for automotive, there were CAN and FlexRay, airspace industry developed SAFEbus and AFDX, and for industrial automation area, there were proposed field-buses, ProfiNet and EtherCAT.

Now, due to the increased requirements regarding bandwidth, the trend is to extend the Ethernet for real-time and safety-critical usage. Examples of Ethernet-based dependable protocols are ProfiNet, EtherCAT, FTT-Ethernet, AFDX, AVB, TTEthernet and TSN. The former two are being known as *Deterministic Ethernet* (DE) [Ste16]. The aim regarding standardization is that protocols with real-time and safety-critical features, such as TSN, are implemented in Commercial Off-The-Shelf (COTS) network devices, and there is a market competition from vendors that would drive the cots down. For example, TSN is proposed to be used in intra-vehicle networks [Hap16], which will further increase cost-pressures, since the automotive market is a cost-conscious mass market. In the future, there will also be interoperability solutions with existing specialized communication protocols, e.g., an EtherCAT to TSN gateway [Eth18] was already launched by EtherCAT Technology Group (ETG) and CAN and FlexRey to TSN gateways are proposed in [SS16].

TSN is not a communication protocol per se, but a collection of sub-standards from IEEE. Initially, IEEE 802.1Q-2005 introduced the prioritized Best-Effort (BE) traffic. In the same year, the IEEE Audio-Video Bridging (AVB) task group is created and has proposed IEEE 802.1Qav [AVB09] to shape the traffic ensuring thus bounded Worst-Case end-to-end Delay (WCD). After that, in 2012, the group was renamed to Time-Sensitive Networking (TSN) whose purpose now is to extend the IEEE 802.1Q Virtual Local Area Network (VLAN) protocol to meet the requirements of time-sensitive applications.

To address the timing constraints, *Shapers* and *schedulers* have been introduced and standardized as: IEEE 802.1Qav AVB (a Shaper for prioritized asynchronous traffic), IEEE 802.1Qbv [TSN15] (an enhancement for TT traffic) and IEEE 802.1Qcr [TSN18b] (an improvement for the asynchronous traffic shaping). The reliability requirements are addressed by Stream Reservation Protocols (SRPs) standardized in IEEE 802.1Qat [IEE10] (a dynamic signaling protocol to reserve network resources for QoS guaranteed streams), 802.1Qcc [TSN18a] (an enhancement for the reservation protocol) and 802.1CB [TSN17] (Frame Replication and Elimination for Reliability).

Table 1.2 gives more details about standards and amendments in TSN. The naming uses the following IEEE conventions: (1) standards are with capital letters; (2) amendments of a standard are with small letters; (3) a Project Authorization Request (PAR) has to be approved to start to develop a standard or amendment; and (4) a standard or amendment that is not published yet is an "ongoing project" and the identifier is prefixed with "p". Furthermore, if a standard is not merged into another standard or is not reviewed or reaffirmed after five years since it has been published, the standard is not considered state of the art anymore.

1.1.3.4 Overview of Traffic Types in Deterministic Ethernet (DE)

In the following paragraphs, we will introduce the traffic types of DE that are of most importance for the work on this thesis, and for the coverage of related work, see subsection 1.2.1. Furthermore, each traffic type (or class) will be presented in terms of *forwarding mechanisms*, *configuration parameters* and *most*

Identifier	Name	Status
802.1Q-2005	Virtual Local Area Networks (VLANs)	Published May 19th 2006
802.1Q-2011	Virtual Local Area Networks (VLANs)	Published Aug. 31st 2011
802.1Q-2014	Bridges and Bridged Networks	Published Dec. 19th 2014
p802.1Q-2018	Bridges and Bridged Networks	PAR approved Sept. 22nd 2016 Standard approved May 7th 2018
802.1AS-2011	Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks	Published Mar. 30th 2011
p802.1AS-Rev	Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks	PAR approved Feb. 16th 2015 Draft 7.0 Mar. 29th 2018
802.1BA-2011	Audio Video Bridging (AVB) Systems	Published Sept. 10th 2011
802.1BA-Corrigendum1	Audio Video Bridging (AVB) Systems	Published Jul. 22nd 2016
802.1CB-2017	Frame Replication and Elimination for Reliability	Published Sept. 28th 2017
802.1Qav-2009	Virtual Bridged Local Area Networks - Amendment: Forwarding and Queuing Enhancements for Time-Sensitive Streams	Published Jan. 5th 2010 Merged into 802.1Q-2014
802.1Qat	Virtual Bridged Local Area Networks - Amendment: Stream Reservation Protocol (SRP)	Published Sept. 30th 2010 Merged into 802.1Q-2014
802.1Qca-2015	Bridges and Bridged Networks - Amendment: Path Control and Reservation	Published Mar. 11th 2016 Merged into p802.1Q-2018
802.1Qbv-2015	Bridges and Bridged Networks - Amendment: Enhancements for Scheduled Traffic	Published Mar. 18th 2016 Merged into p802.1Q-2018
p802.1Qcc	Stream Reservation Protocol (SRP) Enhancements and Performance Improvements	PAR approved Oct. 21st 2013 Standard approved Jun. 14th 2018
p802.1Qcr	Bridges and Bridged Networks - Amendment: Asynchronous Traffic Shaping	PAR approved Jun. 30th 2016 Draft 0.5 Jun. 12th 2018

Table 1.2: TSN standards and amendments

suitable timing requirements. In DE-based networks, the messages may have to be fragmented into several frames, if a message size is larger than the Ethernet Maximum Transmission Unit (MTU) of 1,500 bytes.

The **Time-Triggered (TT)** class is specified in TTEthernet [SAE11]. TT frames are sent and received at predefined points in time that are stored in static *schedule tables*. Along a route, network devices are storing TT traffic in *buffers*, forwarding a frame only when it is scheduled. In TTEthernet the TT traffic has the highest priority and guarantees bounded delay and jitter.

TT is also used in TSN. The main difference between TTEthernet and TSN is that in TTEthernet the shaping and scheduling are done on a *per-message* basis, when in TSN is on a *per-queue* basis. The TT queues are shaped by Time-Aware Shaper (TAS) specified in IEEE 802.1Qbv. With the introduction of TAS a gate is associated to each outgoing queue. So far, in TSN, TT traffic has the highest priority. Therefore, the queued up TT frames are eligible for transmission when: (1) there are no other frames on transmission and (2) the gate is open. *Gate Control List (GCL)* stores the times when the gates of associated queues are open and closed. TSN enables TT traffic with low delay and jitter if (i) the clocks of all network devices that are forwarding TT traffic are synchronized and (ii) GCLs are carefully synthesized to schedule forwarding of TT messages in a route from source to destinations. "Carefully synthesized" means that GCLs must satisfy the frames non-interleaving constraint introduced in [CSCS16] and [SCS18], this making GCLs equivalent with schedule tables in TTEthernet.

Audio-Video Bridging (AVB) is one of the asynchronous traffic types in TSN. For AVB class the Credit-Based Shaper (CBS) is specified in IEEE 802.1Qav and IEEE 802.1BA [AVB11] which defines the configuration parameters for AVB-aware network devices. In TSN, frames of AVB messages are queued up in network devices outgoing ports and forwarded when the queue is eligible for transmission. CBS purpose is to prevent the starvation of lower priority queues. Therefore, a non-empty AVB queue is eligible for transmission when: (1) there are no other frames on transmission, (2) there are no higher priority queues eligible for transmission and (3) the credit is non-negative. When an AVB queue is transmitting the credit decreases with *sending slope* and when a non-empty queue waits for transmission the credit is increased with *idle slope*. *Idle and Sending slopes* are configuration parameters defined in IEEE 802.1BA with the *idle slope* representing a fraction of link speed and the *sending slope* being the difference between link speed and *idle slope*. Furthermore, IEEE 802.1BA defines the latency math to compute the upper bound delay for an AVB message.

The **Rate-Constrained (RC)** is the asynchronous traffic class of TTEthernet defined in Avionics Full-DupleX switched-Ethernet (AFDX) [Aer09]. Each RC message is shaped along its route in order to impose an upper limit for its bandwidth. A RC message has associated a (1) *Bandwidth Allocation Gap (BAG)*, that represents the minimum time interval allowed between two consecutive frames of the message, and (2) a *maximum frame size.* The *BAG* and *maximum frame size* are the configuration parameters used to regulate the RC traffic. Therefore, RC traffic type provides guaranteed bandwidth and has bounded delay.

1.2 Communication-Related Design Tasks

The engineering of large-scale complex systems is enabled by the use of Model-Based Systems Engineering (MBSE) methodologies [Est07], which provide Computer-Aided Design (CAD) support for several design tasks. [PGPE11] outlines the most important tasks used to design a safety-critical system.

In this thesis, we focus on communication-related design tasks, such as architecture selection and communication synthesis. Note that we will use the term *network planning and design* instead of "architecture selection". As Pop et al. [PGPE11] show the communication synthesis for a TT-based architecture is focused on *scheduling.* For a mixed-criticality system, as highlighted in [PRCS16], the communication synthesis, in addition to scheduling, consists also of *routing* and *packing and fragmenting.* Moreover, this thesis addresses also design tasks such as *priority and traffic class assignment* [PRCS16] and *bandwidth allocation.*

The CAD support is implemented as optimization tools that perform *design space exploration* to search for good quality solutions—most of the time, finding the optimal solution is computationally intractable. Hence, in this thesis, we use heuristic- and metaheuristic-based strategies for design-space exploration [BK14]. To *evaluate* the solutions visited during the search, we are defining context-specific cost and quality functions, that employ timing analysis methods, see each paper included.

The *network planning and design* means to design the logical topology of a network, i.e., to decide the number of network switches and cables and how to interconnect them. The *priority and traffic class assignment* deals with deciding, depending on message requirements and resources availability, the appropriate traffic class and priority for each message. To ensure the timing and dependability requirements, DE-based systems use different traffic priorities and traffic classes.

Grouping multiple smaller messages into one frame and splitting a message into multiple smaller frames is referred as *packing and fragmenting*. The task of *routing* supposes to select for each message from the network the tree on which it will be forwarded; the cycle-free connected structure that connects the message source with the message destination(s).

The *scheduling* design task deals with deriving the schedule tables. The scheduled (or TT) traffic is sent and received at predefined points in time which are stored in the so-called schedule tables. *Bandwidth allocation* means to impose a bandwidth upper bound for a message, traffic priority or traffic class. This mechanism prevents non-scheduled or lower priority traffic to wait at infinity on network devices egress ports. Therefore, *bandwidth allocation* helps the non-scheduled traffic by reducing its WCD. Thus, we can say that *bandwidth allocation* is equivalent to the *scheduling* for the non-scheduled traffic.

1.2.1 Related Work

Network planning and design: The problem of network planning and design is well-studied in the literature. For example, it has been addressed in [KRD02] for Industrial Ethernet. For TTEthernet-based networks, the problem of network planning and design is addressed in [TSPM15].

The fault-tolerant network design for DE-based systems is presented as a problem of joint topology and routing design only in [GTSP15] for TTEthernet and in Paper A chapter 2, included in the thesis, for TSN.

Traffic class assignment: The problem of traffic class assignment for TTEthernet-based networks, briefly presented in [GP16], is similar to the problem of priority assignment. For example, in [HSF14] researchers propose a solution for the priority assignment problem of RC messages in AFDX. The method is based on the optimal priority assignment strategy (usually used in the context of real-time tasks) and it assigns higher priorities to the messages with tighter timing constraints.

In the context of TSN the problem of traffic class (or type) assignment addressed in Paper B chapter 3, included in the thesis, is similar to the one for TTEthernet. For more details about differences between traffic classes and configuration parameters in TTEthernet and TSN the reader is redirected to the Paper B chapter 3. Furthermore, also in the context of TSN, the problem of priority assignment is tackled for Asynchronous Traffic Shaping (ATS) traffic in [SS17]. The work in [SS17] searches for the minimum number of queues that are required to guarantee the timing requirements of ATS traffic.

Packing and fragmenting: The problem of frame packing is widely addressed in the literature in [ASBCH13, AMF12, PEP05, SN06], for example. For safety-critical avionics systems based on AFDX networks, multiple messages are packed into one frame, based on (1) their timing availability [AMF12] and (2) their sources and destinations [ASBCH13]. Researchers in [MAR08] present the problem the other way round; in a preemptive TTEthernet context TT messages are split into smaller frames in order to reduce the preemption time of lower priority traffic. An integrated approach though is addressed in [TSPS14], where both packing and fragmenting are used together with routing and scheduling for TTEthernet-based systems to ensure the timing requirements of critical traffic.

Routing: The problem of tweaking the routes in order to improve the schedulability for TSN-based systems has also been addressed. There are solutions based on meta-heuristics and Integer Linear Programming [Lau16] and [Nay17].

The joint problem of routing and scheduling is also well investigated for TSN networks. For example, to solve the joint routing and scheduling problem researchers in [SDT$^+$17], [PTO18] and [ZP18], are proposing an ILP, a List Scheduling-based heuristic and a GRASP (Greedy Randomized Adaptive Search Procedure) meta-heuristic, respectively. All solutions listed previously are using the scheduling and routing to improve only for TT traffic. However, only Paper C chapter 4 included in the thesis addresses the problem of routing and scheduling for mixed-criticality applications, i.e., it searches for the network configuration where the timing requirements of both TT and AVB messages are satisfied.

Scheduling: Synthesis of schedule tables for DE networks is a well-studied problem in literature. For example, for TTEthernet there are solutions to the scheduling problem in [Ste10], for the mixed-criticality applications in [Ste11] and for large systems in [PSRH15]. In TSN the researchers addressed the scheduling problem in [CS16] for joint tasks and messages scheduling and in [CSCS16, PRCS16, RP17, SCS18] for pure messages scheduling by using different searching methods, such as list heuristics, GRASP-based meta-heuristics, satisfiability-modulo theories or Integer Linear Programming (ILP) based methods. On the other hand in [DN16] the scheduling is addressed in the context of AVB-ST networks. The scheduling for mixed-criticality applications over TSN networks is addressed in [GP18].

Bandwidth allocation: Similar with packing for AFDX-based networks, the strategy of messages aggregation is used, i.e., in addition to packing is performed the selection of shaping parameters, such as Bandwidth Allocation Gap (BAG). In [ASBCH13] before packing and routing researchers propose a method to decide the optimal allocated bandwidth for an RC message. To reduce the wasted bandwidth authors in [MLC15] propose a heuristic method to decide the BAG of an aggregated message and the phase and on-source backlog time of an individual aggregating message.

Due to the per-queue shaping of AVB traffic, that has a great impact on the WCD of AVB messages, the problem of bandwidth allocation is even more investigated for TSN networks. For example, [MVNB] proposes for AVB custom-classes a method to decide the minimum idle slope such that the timing requirements of all AVB messages are met and the end-to-end delay of BE traffic is maximized. Researchers in [HZL17] present a method to determine, based on the required bandwidth of messages, the per-port idle slope, such that the WCD of messages having different AVB classes are minimized.

1.3 Thesis Overview and Contributions

During the course of my Ph.D. studies I have published and submitted the following publications:

- Voica Gavrilut and Paul Pop, **"Traffic Class Assignment for Mixed-Criticality Frames in TTEthernet"**, accepted and presented at the Euromicro Conference on Real-Time Systems (ECRTS) workshop: Real-Time Networks (RTN), 2016. [GP16]

- Voica Gavrilut, Bahram Zarrin, Paul Pop and Soheil Samii, **"Fault-Tolerant Topology and Routing Synthesis for IEEE Time-Sensitive Networking"**, accepted and presented at the Real-Time Networks and Systems (RTNS), 2017. [GZPS17]

- Voica Gavrilut and Paul Pop, **"Scheduling in time sensitive networks (TSN) for mixed-criticality industrial applications"**, accepted and presented at the Workshop on Factory Communication Systems (WFCS), 2018. [GP18]

- Voica Gavrilut and Paul Pop, **"Traffic Type Assignment for TSN-based Mixed-Criticality Cyber-Physical Systems"**, submitted to the ACM Transactions on Cyber-Physical Systems (TCPS). [GPew]

- Voica Gavrilut, Luxi Zhao, Michael L. Raagaard and Paul Pop, **"AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN"**, accepted in the IEEE Access journal. [GZRPss]

The following articles have been included in this thesis: [GZPS17] as Paper A (chapter 2), [GPew] as Paper B (chapter 3), and [GZRPss] as Paper C (chapter 4). This thesis focuses on the following design tasks: (1) network planning and design, presented in Paper A chapter 2, (2) traffic type assignment, presented in Paper B chapter 3, (3) traffic routing, presented in all papers, Paper A, Paper B, and Paper C chapter 4, (4) traffic scheduling Papers B and C, and (5) bandwidth allocation, Paper B.

In the following, we briefly describe each of the three papers and their contributions.

1.3.1 Paper A: Fault-tolerant topology and routing synthesis for IEEE Time-Sensitive Networking

Because this thesis targets safety-critical applications, Paper A considers that the routing of messages is determined statically at design time. However, non-critical messages can use dynamic route and bandwidth reservation mechanisms provided by TSN.

Safety-related systems have to be developed according to certification standards, such as ISO 26262 for the automotive area or IEC 61508 used for industrial applications. Considering the current certification practice, we assume that the engineer will specify for each application, depending on its criticality, the required Redundancy Level (RL). This means that critical messages have to be routed through RL redundant disjoint routes.

Therefore, in Paper A, we consider the set of ESes and of critical messages, including their RL, as given, and the focus is on determining a minimum cost fault-tolerant network architecture which can guarantee the timeliness and dependability requirements of the critical messages. For this problem initially, we derived the fault-tolerant topology with the maximum cost. After applying the routing strategy the network switches are downgraded as much as possible and the unused links and switches are removed.

For routing, we proposed three searching strategies: (i) A heuristic strategy, called Topology and Routing Heuristic (TRH), which is based on the breadth-first search algorithm. (ii) The second strategy is to model the routing problem using constraint programming. The method is denoted Topology and Routing Optimization (TRO) and is an exhaustive search where the variable domains are reduced by the mean of constraints. (iii) Finally, we have proposed a metaheuristic approach, based on the Greedy Randomized Adaptive Search Procedure (GRASP).

Contribution. To the extent of our knowledge, Paper A was the first one which addressed, in the context of TSN, the *topology selection* for fault-tolerant applications and *routing* for multicast redundant messages.

1.3.2 Paper B: Traffic type assignment for TSN-based mixed-criticality cyber-physical systems

Similar to the debate in real-time systems between time-triggered and event-triggered implementations there is no agreement on the appropriate traffic type for the messages of mixed-criticality applications. Paper B focuses on applications with different timing requirements such as Hard Real-Time (HRT), Soft Real-Time (SRT) and applications that are Not time-Critical (NC). In this context, HRT messages have hard deadlines, whereas for SRT messages we capture the Quality-of-Service (QoS) using soft deadlines and "utility functions". We assume that the network engineer assigned for each SRT message such a utility non-increasing function which shows how service degrades when the message is received after its soft deadline. Therefore, the work in *paper B* addresses the problem of "traffic type Assignment" for mixed-criticality messages in TSN.

As input to this problem, we have (1) the network topology and (2) the set of HRT and SRT messages, including their routes. The aim is to map a traffic type to each critical message and to synthesize the schedule tables for TT traffic such that all HRT messages are schedulable and the overall utility of SRT messages is maximized. To solve this problem, we implemented a Tabu Search metaheuristic strategy.

Contribution. Paper B is the first one, to our knowledge, which addressed the problem of traffic type assignment for mixed-criticality applications on TSN.

1.3.3 Paper C: AVB-aware routing and scheduling of time-triggered traffic for TSN

Paper C addresses real-time applications (implemented using both TT and AVB traffic types) running on TSN-based distributed architectures. TT has the highest priority and relies on mechanisms for forwarding queued frames from a specific queue of an egress port at precise points in time. This is implemented by blocking the other queues and relies on static schedule tables, called Gate Control Lists (GCLs), for deciding when to open and close each queue. By synthesizing carefully the GCLs, TT messages can have low end-to-end latency and low jitter. AVB traffic is intended for applications that require bounded end-to-end latencies but has a lower priority than TT traffic. Thus, in this work, the focus is on determining the routes and GCLs for TT traffic, such that both TT and AVB messages are schedulable.

As input to the problem in Paper C, we have the network topology and the set of TT and AVB messages, and the goal is to determine the routes and the schedule tables of TT messages such that all messages are schedulable. In TSN the synthesis of schedule tables means: (1) to map TT messages to TT queues and (2) to compute the GCLs for TT queues. For solving this problem we used an integrated heuristic-metaheuristic strategy: the K-Shortest Path (KSP) based method generates multiple routing alternatives and for each routing alternative the Greedy Randomized Adaptive Search Procedure (GRASP) computes the schedule tables.

Contribution. To the best of our knowledge, Paper C was the first one to address the joint problem of routing and scheduling for mixed-criticality traffic on TSN-based networks.

Chapter 2

Paper A: Fault-Tolerant Topology and Routing Synthesis for IEEE Time-Sensitive Networking

Time-Sensitive Networking (TSN) is a set of IEEE standards that extend Ethernet for safety-critical and real-time applications. TSN is envisioned to be widely used in several applications areas, from industrial automation to in-vehicle networking. A TSN network is composed of end systems interconnected by physical links and bridges (switches). The data in TSN is exchanged via streams. We address safety-critical real-time systems, and we consider that the streams use the Urgency-Based Scheduler (UBS) traffic-type, suitable for hard real-time traffic. We are interested in determining a fault-tolerant network topology, consisting of redundant physical links and bridges, the routing of each stream in the applications, such that the architecture cost is minimized, the applications are fault-tolerant (i.e., the critical streams have redundant disjoint routes), and the timing constraints of the applications are satisfied. We propose three approaches to solve this optimization problem: (1) a heuristic solution, (2) a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic, and (3) a Constraint Programming-based model. The approaches are evaluated

on several test cases, including a test case from General Motors Company.

2.1 Introduction

Many safety-critical real-time applications, following physical, modularity or safety constraints, are implemented using distributed architectures, composed of heterogeneous processing elements (PEs) embedded in "smart" devices which are interconnected in a network. A large number of communication protocols have been proposed for embedded systems. However, only a few protocols are suitable for safety-critical real-time applications [Rus01]. In this paper, we are interested in the protocol colloquially known as Time-Sensitive Networking (TSN) [TSN12]. TSN is used in several application areas, from industrial automation to automotive architectures. For example, in the automotive area, fault-tolerant TSN networks are envisioned in future autonomous driving architectures, since they have the bandwidth requirements to integrate traffic from multiple sensors and the dependability required for autonomous driving.

Ethernet [IEE12], although it has low cost and high speed, is known to be unsuitable for real-time and safety-critical applications [Dec05]. For example, in half-duplex implementations, frame collision is unavoidable, leading to unbounded transmission times. [Dec05] presents the requirements for a real-time network and how Ethernet can be improved to comply with these requirements. Several real-time communication solutions based on Ethernet have been proposed. [SG12] and [CRE$^+$12] describe and compare several of the proposed Ethernet-based real-time communication protocols.

TSN [TSN12] is a set of sub-standards which extend the IEEE 802.1 standards (for switched Ethernet networks) for safety-critical and real-time applications. First, IEEE 802.1Q-2005[1] introduced support for prioritizing the Best-Effort (BE) traffic in order to improve Quality of Services (QoSs). Following this, the IEEE Audio-Video-Bridging (AVB) Task Group was formed to develop another set of enhancements, namely IEEE 802.1BA known as AVB. This standard introduces two new shaped AVB traffic-types, with bounded *Worst-Case end-to-end Delays* (WCDs). In 2012, the AVB Task Group was renamed to IEEE 802.1 Time-Sensitive Networking Task Group to reflect the shifted focus onto further extending the protocol towards safety-critical and time-sensitive transmissions, and has introduced new traffic types such as Time-Triggered (TT) [TSN15] and Urgency-Based Scheduler (UBS) [SS16].

[1]We will not provide references for all standards, but these can be easily found based on their names.

In this paper, we are interested in safety-critical real-time applications. We consider that the application messages use the Urgency-Based Scheduler (UBS) traffic-type IEEE 802.1Qcr [TSN18b]. UBS is an asynchronous traffic scheduling algorithm, which gives low delay guarantees while maintaining a low implementation complexity. It also provides a temporally-composable timing analysis, see section 2.4 and [SS16] for more details. Compared to the TT traffic type, UBS does not require schedule tables, which can be difficult to create, and compared to AVB, UBS guarantees lower latencies and has a simpler and faster timing analysis. However, although we consider UBS in this paper, our approach can handle any combination of traffic types, as long as a timing analysis is available. The choice of traffic type depends on the characteristics of the applications, and the problem of determining the appropriate traffic types has been addressed in [GP16] for mixed-criticality traffic in TTEthernet.

TSN is highly suitable for applications of different safety criticality levels, as it offers spatial separation for mixed-criticality traffic through the concept of Virtual Local Area Network (VLAN), as well as temporal separation through the various traffic type mechanisms. A TSN network is composed of End Systems (ESes) interconnected by physical links and Network Switches, also known in TSN as Bridges (Bs). The links are full duplex, and the network can be multi-hop, see section 2.2 for the architecture model. The data in TSN is exchanged via *streams*, see section 2.3 for the application model. Because we target safety-critical applications, we consider that the routing of streams is determined statically at design time. However, non-critical streams can use dynamic route and bandwidth reservation mechanisms provided by TSN, see [IEE10] and [TSN18a].

We are targeting safety-related systems, which have to be developed according to certification standards; for example, IEC 61508 is used in industrial applications, ISO 26262 is for the automotive area, whereas DO 178C refers to software for airborne systems. Considering the current certification practice, we assume that the engineer will specify for each application, depending on its criticality, the required *Redundancy Level* (RL). At the level of the network topology, this translates into requirements for redundant disjoint routes between the ESes involved in the communication. Thus, if a physical link or a bridge will fail, the other routes can still deliver the information by the deadlines. The current approach in such a situation is to use hardware redundancy at the network level and replicate the complete network, as discussed by [ART14] for an avionics network. Such a solution may not be scalable in terms of weight, space, and resource efficiency for application areas where functions have varying redundancy requirements.

In this paper, our focus is on determining a low-cost fault-tolerant network architecture, which can guarantee the safety and real-time requirements of the applications. We assume that the applications and ESes are given and that the designer has established the redundancy levels, depending on the criticality of the applications. We are interested in determining a fault-tolerant network topology, consisting of redundant physical links and bridges, the routing of each stream in the applications, such that the architecture cost is minimized, the applications are fault-tolerant (i.e., the critical streams have RL redundant disjoint routes), and the timing constraints of the applications are satisfied.

Contributions: This is the first time, to our knowledge, that the problems of (i) topology synthesis and (ii) routing of time-sensitive traffic have been addressed for TSN. We propose three strategies to solve these problems: (1) a fast heuristic solution, (2) a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic that finds good quality solutions in a reasonable time, and (3) a Constraint Programming-based model that searches for the optimal solution.

The paper is organized as follows. The next section presents the related work. section 2.2 and section 2.3 present the topology architecture and traffic models used in the paper. The concepts related to TSN relevant for our paper are presented in section 2.4. The problem formulation is presented in section 2.5 and illustrated with a motivational example in subsection 2.5.1. The proposed optimization strategies are presented in section 2.6, and section 2.7 presents our experimental evaluation.

2.1.1 Related Work

Researchers have started to address the analysis and optimization of "Deterministic Ethernet" (DE) protocols, such as TTEthernet, Industrial Ethernet and TSN. The problem of determining the network topology, i.e., the number of bridges and their interconnection via physical links and to the end systems, is called *network planning and design*. This problem has been addressed for DE in the context of Industrial Ethernet [KRD02] and TTEthernet in aerospace [TSPS14].

In the telecommunications area, there is a lot of work on network reliability and redundancy optimization. An annotated overview of system reliability optimization, which covers also network reliability is presented in [KP00]. In [KS06], the authors present the latest research results in network reliability optimization. Several network reliability measures have been proposed in the literature, such as connectivity, resilience and performability. Researchers have proposed several approaches to the optimization problem, including heuristics, metaheuristics and exact solutions based, for example, on mathematical programming [KS06].

However, these results cannot be applied directly to DE. One of the basic assumptions of earlier works on network reliability optimization is that once a fault is detected, the network will reconfigure itself to avoid the fault. That is, new routes will be found for messages. In the case of DE the routes for safety-critical applications are typically *static*: they are loaded into the end systems and network switches at design time, and it is not possible to change the routing dynamically, at runtime. In this context, researchers have proposed a fault-tolerant topology selection for TTEthernet [GTSP15]. However, for non-critical applications, runtime reconfiguration, including routing, is a relevant problem.

Routing optimization is a well-studied subject where Wang et al. [WH00] and Grammatikakis et al. [GHKS98] provide excellent overviews of the different centralized and distributed routing algorithms. Researchers have also addressed routing in safety-critical systems [HKGF09], [PA04]. For ARINC 664p7, Al Sheikh et al. [ASBCH13] proposed an approach to find the optimal routes in ARINC 664p7 networks using Mixed Integer Linear Programming. Tămaş-Selicean et al. [TSPS14] have used a Tabu Search-based metaheuristic to, among other things, optimize the routing of the RC traffic type to minimize the WCDs in TTEthernet systems.

Regarding routing in TSN, AVB flows are typically established at runtime using the *Stream Reservation Protocol (SRP)* [TSN18a] where either the *Rapid Spanning Tree Protocol* (RSTP) or *Shortest Path Bridging* (SPB) are used to determine the routing. The future enhancements around TSN will support more sophisticated runtime routing algorithms, and the possibility to also determine the routes offline. Researchers have proposed an offline routing optimization approach for AVB in [Lau16]. However, routing for time-sensitive traffic types such as TT and UBS has not been addressed previously.

2.2 Architecture Model

We model the architecture, which is a TSN network as an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where the vertices (or nodes) $\mathcal{V} = \mathcal{ES} \cup \mathcal{B}$ denote the set of all End Systems (ESes) and network switches, usually denoted in TSN as Bridges (Bs), respectively. The edges $\mathcal{E}$ are the full-duplex physical links interconnecting the ESes and Bs. An ES can be of several types. For example, in automotive architecture, an ES is typically an Electronic Control Unit (ECU) composed of a CPU, memory, and I/Os. However, an ES could also be an intelligent sensor such as video camera, radar, or LiDAR. All ESes regardless of their type have a Media-Independent Interface (MII) connector which is a full-duplex digital interface to connect the ES to the network. Figure 2.1 shows an example network with 4 ESes and 4 Bs.

In this paper, for the given set of ESes, we determine the set of bridges, $\mathcal{B}$ to be used and the physical interconnections. We assume that the system engineer provides a network component library $\mathcal{L}$ including a set of bridge types $\mathcal{BT}$ and a set of physical link types $\mathcal{LT}$. Such a library will be defined based on the TSN bridges and physical links available on the market and suitable for the application area considered. For example, TTTech Computertechnik AG and Infineon Technologies AG provide TSN bridges, for the automotive area, with different number of ports and different physical layer technologies, such as the IEEE 802.3 standards for automotive 100 Mbit/s and 1 Gbit/s Ethernet. Our approach is general and can be applied in several areas from automotive to industrial automation.

The library is defined as $\mathcal{L} = (\mathcal{BT}, \mathcal{LT}, BC)$, where $\mathcal{BT}$ is a set of bridge types, and $\mathcal{LT}$ is the set of physical link types. In general, an ES can be connected to any bridges. However, in practice, there can be constraints that limit the type of bridges and physical links that can be used by an ES. For example, a video camera could impose a limit on the fit of the bridge (due to the fact that the ES and bridge are packaged into the same electrical component), such that it fits together with the camera in the desired location in the vehicle. Therefore, there are packaging constraints for some ESes that limit the network topology synthesis. We capture the bridge constraints with the function BC, which is a mapping from an End System ES_i to the limited set of the bridge and physical link types that can be used by ES_i. In Figure 2.1 all bridges have assigned the bridge type bt_2, which has 1 internal and 3 external ports.

We use two functions to specify the type of the bridges and the physical links used within the architecture network. The first function $BT : \mathcal{B} \mapsto \mathcal{BT}$, specifies the type of a bridge, e.g., $BT(B_1) = bt_2$ in Figure 2.2. The other function

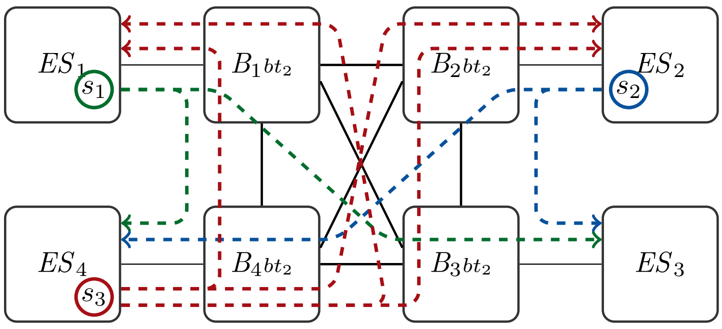


Figure 2.1: Architecture model example

Id	Cost	No.int. ports	No.ext. ports
bt_1	2	1	2
bt_2	8	1	3
bt_3	10	2	2

Id	Cost	Speed Mbit/s	Int./Ext.
lt_1	7	100	Ext.
lt_2	1	1000	Int.

Table 2.1: Example library $\mathcal{L}$

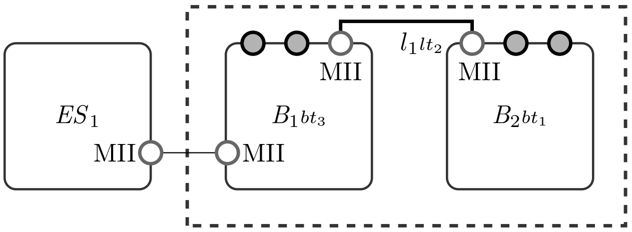


Figure 2.2: Chaining bridges

$LT : \mathcal{E} \mapsto \mathcal{LT}$, specifies the type for each link in the network topology, e.g., $LT(l_1) = lt_1$. We represent the monetary cost of a bridge, ES, and physical link as $Cost$, e.g., $B_1.Cost = 8$. We denote the transmission rate of a physical link as $speed$, e.g., $l_1.speed = 100$ Mbit/s, the connectivity type of a physical link as lct, e.g., $l_1.lct =$ Ext, and the number of int. and ext. ports of a bridge as $noIntPorts$ and $noExtPorts$, e.g., $B_1.noIntPorts = 1$.

Similar to the network engineering practice, we will allow the "chaining" of several bridges to construct a new type of bridge, that has more ports, hence supporting more connections. Figure 2.2 shows an automotive ECU with a microcontroller ES_1 connected to a bridge that is built from chaining two bridges B_1 and B_2 of types bt_3 and bt_1, respectively.

We distinguish between two types of connections: internal links, which are between the MIIs of ESes and bridges, and external links, which connects two bridges using a physical connector, colloquially known as PHY. A PHY consists of a physical digital to analog converter, as well as filters to support the bit rate with the required signal qualities within the operating environment, and a connector to the wiring, for example. Both internal and external links are physical links denoted as l_i, $l_i \in \mathcal{E}$, which are bidirectional. We call a link connectivity ESes to bridges an internal link because there are application areas, e.g., automotive, where the bridge is integrated with the ES on the same board, so the internal link is a connection on the PCB between the pins of the microcontroller/sensor and the pins of the TSN bridge. Such an internal link is more reliable compared to an external link, which is susceptible to PHY connector failures, see subsection 2.3.1 for the fault model.

A dataflow link (DL) dl_j represents a directed connection on a physical link l_i, $(ES_1 - B_1)$ from Figure 2.1 for example. A dataflow path (DP) dp_k is a sequence of interconnected DLs. Such a path in Figure 2.1 is $[(ES_1 - B_1), (B_1 - B_4), (B_4 - ES_4)]$. The set of all DLs is denoted with $\mathcal{DL}$ and the set of all DPs is $\mathcal{DP}$.

2.3 Application Model

The safety-critical real-time applications are modeled as periodic tasks distributed on the ESes. Our application model captures the communication among tasks via streams. A stream is denoted as s_i and the set of all streams is denoted $\mathcal{S}$. Streams may be multicast, so each stream s_i has a source $s_i.src$, which is an ES, and has one or multiple destinations ESes $s_i.dests$. The messages transmitted in a stream may be split into several packets, and each packet is wrapped in an Ethernet *frame*. The messages of a stream may have to be fragmented into several packets, if their length is larger than the Ethernet Maximum Transmission Unit (MTU) of 1,500 bytes. The problem of message fragmenting and frame packing is orthogonal to our problem, and has been addressed in the context of Deterministic Ethernet [TSPS14].

We use the leaky bucket traffic model in this paper, see [SS16] for more details, which means that each stream s_i is characterized by a burstiness $s_i.B$, which represents the maximum amount of data that can be transmitted at once, and a leak rate $s_i.R$. Our application model can accept any type of streams which satisfy the leaky bucket constraint, i.e., for a stream s_i the total amount of data w_i accumulated on a duration d is bounded by $w_i(d) \leq s_i.B + d \cdot s_i.R$. Each frame of a stream $s_i \in \mathcal{S}$ has a deadline $s_i.D$ by which the frame has to arrive at its destinations, relative to the releasing of each frame. The advantage of such a traffic model is its versatility: it can model strictly periodic streams with fixed size frames, sporadic streams, as well as variable sized frames useful for multimedia data and large data payloads that need to be transmitted in back-to-back frames, see [SS16] for more details. For example, a strictly periodic stream s_i, with a packet size $s_i.size$ a period $s_i.T$ and an absolute deadline $s_i.deadline$ by which the message need to be delivered, can be modeled with the leaky bucket model as: $s_i.R = s_i.size/s_i.T$ and $s_i.B = s_i.size$ and $s_i.D = s_i.deadline$. An aperiodic stream with a maximum allowed amount of data $s_i.size$ exceeding MTU and a minimum inter-arrival time, which is denoted $s_i.T$, can be similarly modeled with all frames on which the streams is fragmented inheriting the stream's relative deadline $s_i.D$.

We model the routing of a stream as a Multicast Tree $mt_s(s_i)$, a directed struc-

Id	Src.	Dests.	B in B	R in ms	D in ms	rl
s_1	ES_1	ES_3, ES_4	150	15	7	1
s_2	ES_2	ES_3, ES_4	100	10	4.5	1
s_3	ES_4	ES_1, ES_2	100	10	4	2

Table 2.2: Application model example

ture with the source as root and destinations as leaves. $\mathcal{MT}_s$ is the set of all multicast trees. Figure 2.1 shows 4 trees. For example, $mt_s(s_1)$ for the stream s_1 from ES_1 to ES_3 and ES_4, has the route $ES_1 - B_1 - [[B_3 - ES_3], [B_4 - ES_4]]$ depicted with a thick green dashed arrow. For each original stream $s_i \in \mathcal{S}$ we denote with $s_i^j, 1 \leq j \leq s_i.rl$ its j^{th} redundant copy, s_i^1 being s_i itself, and $s_i.rl$ is the stream's redundancy level, see subsection 2.3.1. The set $\mathcal{S}^\star$ denotes the set of all streams and their redundant copies. Table 2.2 shows an example application model, with s_3 having a redundancy level of 2 and the other two streams not being fault-tolerant. The routes for the streams listed in Table 2.2 are depicted in Figure 2.1.

Due to the delays implied by path recovery in case of a physical fault, in this paper we proposed a network configuration where the routes are static: they are determined and loaded into the end systems and bridges at design time. For the non-critical streams the routes can be determined also dynamically, e.g., using the TSN sub-standards as 802.1Qat or 802.1Qcc.

As discussed in the introduction, we assume that each stream uses the UBS traffic-type. UBS allows the definition of a non-unique stream priority $s_i.priority$, which can change at each hop. The assignment of priority is an interesting optimization problem. However, in this paper we assume that the priorities are given as input by the system engineer, and without loss of generality we assume that the priority is fixed for a stream. For example, the priority for a stream $s_i \in \mathcal{S}$ could be defined by the ratio $s_i.B/s_i.R/s_i.deadline$, thus the stream with higher burstiness, lower leak rate and lower deadline has a higher priority. In our example from Table 2.2 we consider that all streams have the same priority.

2.3.1 Fault Model

Critical streams have to deliver their data even in the case of permanent failures. As mentioned, we assume that the system engineer provides for each stream s_i the required redundancy level (RL) $s_i.rl$, which means that the stream s_i has to be routed on $s_i.rl$ disjoint routes, such that the failure of any $RL - 1$ routes does not result in communication failure of the stream.

Our model is complementary to common probabilistic models of diagnostic and reliability requirements, such as MTTF targets established for various safety integrity levels in the automotive functional safety standard ISO 26262. New application areas, such as fail-operational autonomous driving systems [On-14], have functional safety requirements that are not currently addressed by ISO 26262. For example, some systems may require that there is no single point failure; absence of dual point failures are also seen in some highly critical applications. This is evident in the failure models considered in recent work in industry standardization (redundant communication paths in Ethernet [TSN17]) and research work on dependable real-time Ethernet [AGRN16] to synthesize robust time-triggered schedules for a given number of maximum link failures. Similar requirements can be seen in some avionics and industrial control applications.

The most common types of permanent hardware failures is the physical connector (PHY) failures [Sie04, Tib13], i.e., the cable terminals are corroded due to vibration and thermal fluctuations. End Systems (microcontrollers, smart sensors) and Bridges (network switches) are less likely to fail [Sie04]. The internal links (MII) are on the PCB (microcontroller and switch are all on the same board), hence an internal link failure would result in an ES failure, from a system perspective.

2.4 TSN Protocol and UBS

In this paper, we consider that the streams are scheduled using the UBS traffic type. UBS has been proposed in [SS16] and, due to its advantages, it is currently being standardized by the TSN Task Group as IEEE P802.1Qcr [TSN18b]. UBS is a Rate-Constrained (RC) class, which means it does not rely on the availability of network clock synchronization (required for the TT traffic-type) or on offline synthesis and coordination of schedule tables. Moreover, due to the leaky bucket traffic model used in UBS (see section 2.3), it does not impose any constraints on the burstiness or leak rate of streams.

The TSN sub-standards are amendments to IEEE 802.1Q, which is the standard for Bridged Virtual Local Area Networks using full-duplex IEEE 802.3 Ethernet. 802.1Q introduced additional content in the Ethernet frame header, including a 3-bit Priority Code Point (PCP) identifying up to 8 priority levels.

In 2.3a, we show the general structure of a 4-port TSN-aware bridge with the following main functionality: traffic policing, switching, traffic shaping and

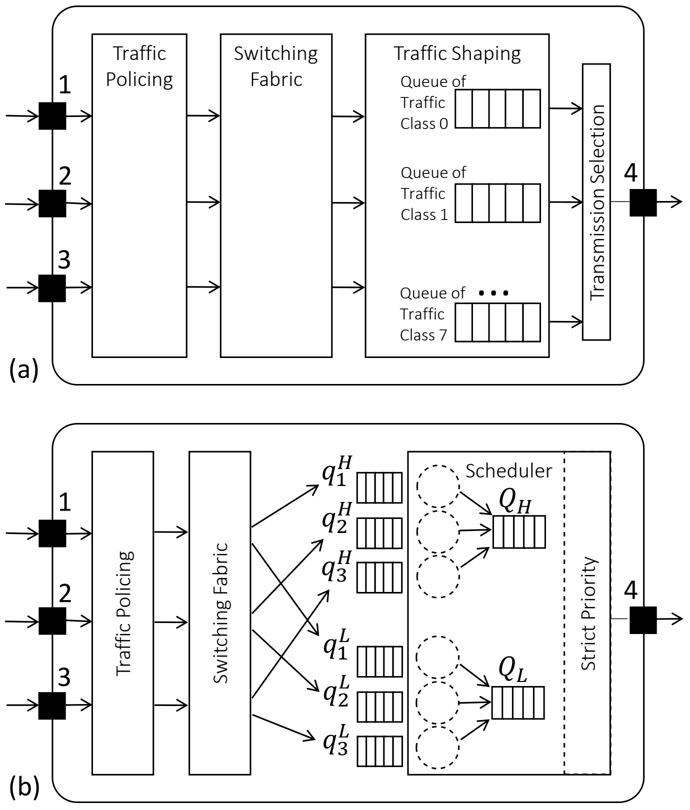


Figure 2.3: (a) Structure of a TSN-aware bridge and (b) UBS shaping for (a)

transmission selection. For presentation purposes, without loss of generality, we show only the ingress portion for the left 3 ports and only the egress portion of the right hand port. On ingress, frames go through a policing engine, which can be used to limit the allowed traffic and its bandwidth (TSN standard P802.1Qci). The switching is aware of each stream's route and forwards incoming frames to one (unicast) or more (multicast) egress ports based on the Address Resolution Logic (ARL) table, which is one of the decision variables of our routing synthesis problem. Each egress port contains a number of queues, each of which is configured to support a traffic type—for example, TT, Credit-Based Shaping (CBS) as in AVB, UBS, or FIFO-like for best-effort traffic. Each queue is configured at a fixed priority (with 8 priority levels available). The transmission selection algorithm selects frames for transmission based on the priority levels and based on whether or not a queue has a frame available for transmission. The traffic shaping, queuing, and transmission selection mechanisms are also implemented by each TSN-aware end system.

Incoming streams to the bridges shown in 2.3a and 2.3b are forwarded to the appropriate egress queue based on the stream identifier (typically the destination MAC address and optionally also VLAN identifier) and the frame priority value (i.e., the PCP). For UBS, each queue is shaped to satisfy the cumulative leaky bucket constraint of the streams mapped to that queue. The transmission selection algorithm then prioritizes the traffic based on queue priorities (discussed in section 2.3 and detailed in [SS16]).

 2.3b shows a detailed view of UBS shaping at the egress port, for the same bridge shown in 2.3a. Incoming UBS streams are statically mapped to the UBS queues q_1^H, q_2^H, q_3^H, q_1^L, q_2^L, and q_3^L, which could be the first six queues in 2.3a (the last two could, for example, be dedicated to noncritical, best-effort traffic), with the rule that frames on different ingress ports are mapped to different queues (for fault isolation purposes). Frames in each UBS queue are shaped to satisfy the leaky-bucket constraint; the shaper is shown with a dashed circle. The purpose of the shaper is to establish whether or not the frame at the head of the queue is eligible for transmission, based on leaky-bucket constraints. Each queue has a fixed priority and it is possible that two or more queues have the same priority, for flow aggregation purposes. We assume that the order of priority levels is preserved through each hop along the route; see section 2.3 for a more detailed explanation and [SS16] for the structure of a general purpose bridge. In our example, the bridge is aware of two priority levels, high (H) and low (L). Queues q_1^H, q_2^H, and q_3^H have the same priority (H) and are, after shaping, therefore merged into the logical FIFO queue Q_H (called *pseudo queue* in [SS16]). Similarly, queues q_1^L, q_2^L, and q_3^L have priority L and are merged into the logical FIFO queue Q_L. Note that Q_H, Q_L, and the strict priority scheduler in 2.3b correspond to the Transmission Selection block of 2.3a. In case the queue priorities are unique, there is no merging into logical queues after traffic shaping.

In the worst-case, in the scheduler of the sending node, a frame of stream s_i is delayed by all streams of higher priorities H, all streams of the same priority $C(i)$ and by the frame of maximum size of a lower priority stream $size_L$. On the receiver side, the frame is delayed, in the worst-case, by the slowest stream (i.e., the stream with highest burstiness). We use the analysis method from [SS16] to check the schedulability of each frame of a stream $s_i \in \mathcal{S}^\star$.

2.5 Problem Formulation

The problem we are addressing in this paper can be formulated as follows. As an input we have (1) the set of end systems $\mathcal{ES}$, (2) the library of components

$\mathcal{L}$, (3) the set of streams $\mathcal{S}$ for which we know the source, destination(s), and timing properties as well as the desired redundancy level $s_i.rl$. We are interested in determining an optimized solution $Sol = (\mathcal{G}, SR)$, where $\mathcal{G}$ is the network architecture and $SR : \mathcal{S}^\star \longmapsto \mathcal{MT}_s$ is a function that specifies the routing expressed as multicast trees for all the streams and their redundant copies, such that the architecture cost is minimized, the applications are fault-tolerant, considering the specified redundancy levels, and the timing constraints of all streams are satisfied.

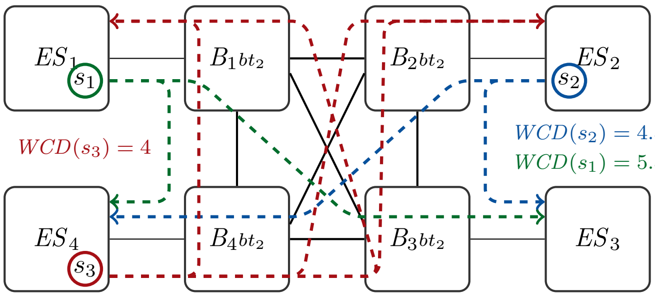
2.5.1 Motivational Example

Let us consider the example from Figure 2.4 where we have 4 ESs , ES_1 to ES_4 and the applications from Table 2.2. As library components we have 3 bridge types, bt_1 to bt_3, with types properties presented in Table 2.1 and where all ESes can be directly connected to all types of bridges. For these examples the physical links have a speed of 100 KBps and a cost of one monetary unit. In Figure 2.4 the gray lines represent the internal links, the thicker black lines the external ones and the colored directed arrows are used for showing the stream routes. We present for each stream its WCD calculated by the analysis in section 2.4. The streams in this example are schedulable if the WCDs are smaller or equal to the relative deadline D from Table 2.2.

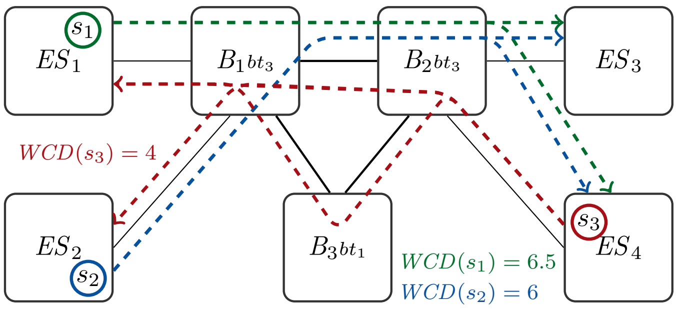
The topology that maximizes redundancy without concern for cost is shown in 2.4a. To obtain this topology, we have connected each ES to its own bridge, and we have introduced full connectivity among the bridges: each bridge is connected to all other bridges. The bridge type is selected from the library such that it accommodates the required ports. The cost of such topology in 2.4a is 42 monetary units. As expected, we can find disjoint redundant routes for s_3, which is fault-tolerant, and all streams are schedulable.

We can reduce the cost to 29 monetary units if we use the topology from 2.4b, which uses 3 bridges (although their individual cost is higher) and fewer physical links. We are able to find disjoint redundant routes for s_3. To determine the routes, in 2.4b we use the shortest path approach. However with this routing, s_2 is not schedulable. Because we are routing both s_1 and s_2 through link $(B_1 - B_2)$, in the worst case s_2 is delayed by frames of s_1 such that the $s_2.D$ is not satisfied.

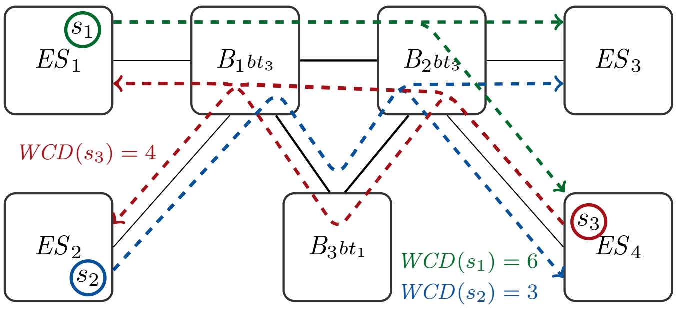
Our approach optimizes both the physical topology and the routing of streams. 2.4c shows the same topology from 2.4b, but where the routes are optimized; counterintuitively, they may now take longer route compared to 2.4b. Here we can see that by routing s_2 through a longer route, namely $ES_2 - B_1 - B_3 - B_2 -$



(a) $cost_A = 42, \delta_t = -1.5$



(b) $cost_A = 29, \delta_t = 1.5, s_2$ unschedulable



(c) $cost_A = 29, \delta_t = -2.5$, all schedulable

Figure 2.4: Motivational example

$[ES_3, ES_4]$ the routes for redundant copies are fully disjoint and all streams are now schedulable.

As we can see from this example, by only optimizing the topology and routing we are able to minimize the cost and guarantee the fault-tolerance and timing constraints of streams.

2.6 Synthesis Strategies

The optimization problem described in the previous section is NP-hard. According to [WH00], any routing problem subject to at least two additive or multiplicative tree constraints is an NP-hard problem. Following the classification from [WH00], our problem can be expressed as a graph optimization problem subject to: (1) link constraint: the capacity of the links should not be exceeded, (2) the number of links adjacent to a vertex should not exceed the node number of ports, (3) the routes of redundant streams are link-disjoint[2] interrelated tree constraint and (4) all streams should be schedulable. Consequently, based on constraints (3) and (4), our optimization problem is NP-hard. To solve this problem, we propose three strategies, (1) a heuristic-based approach, further called Topology and Routing Heuristic (TRH), see subsection 2.6.2, (2) a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic, see subsection 2.6.3, and (3) a Constraint Programming-based strategy, further referred as Topology and Routing Optimization (TRO), see subsection 2.6.4. In order to evaluate the visited solutions, all strategies use the cost function defined in subsection 2.6.1.

2.6.1 Cost Function

A solution is evaluated using the following cost function:

$$cost_T(Sol(\mathcal{G}, SR)) = \wp_{sched} * \delta_t(SR) + \wp_{topo} * cost_A(\mathcal{G}) \tag{2.1}$$

Where the first term is used to check if the solution is schedulable, the second term captures the architecture cost, and $\wp_{sched}$ and $\wp_{topo}$ are constant weights. In order to be able to aggregate the two terms, we normalize the two values. For both, δ_t and $cost_A$, the minimum and maximum values are computed and the actual values scaled in the range $(0, 1]$. To increase the probability of finding a

[2]The number of commonly used links should be 0. For example, if $R_1 = G(V_1, E_1)$ and $R_2 = G(V_2, E_2)$ represent the multicast trees of two redundant copies of the same stream, then $E_1 \cap E_2 = \emptyset$

solution we relaxed the schedulability constraint adding it as a soft constraint, i.e., as a highly penalized part of the cost function. In order to distinguish among the topologies of similar costs we are going for those solutions which: (a) are schedulable and (b) once they are schedulable, they should reduce the WCDs, see 2.4c. Therefore, the weighted penalty for the first term $\wp_{sched}$ is significantly higher than the architecture penalty $\wp_{topo}$.

The monetary cost of the network architecture is the sum over the cost of all bridges and all physical links in the topology $\mathcal{G}$:

$$cost_A(\mathcal{G}(\mathcal{V}, \mathcal{E})) = \sum_{v \in \mathcal{V}} v.Cost + \sum_{e \in \mathcal{E}} e.Cost \tag{2.2}$$

The degree of schedulability δ_t represents the amount of tardiness with which all streams are arriving after their relative deadline, having a negative tardiness for a schedulable stream. We define $\delta_t(SR)$ as follows:

$$\delta_t(SR) = \begin{cases} \text{if at least one stream is not schedulable} \\ \displaystyle\sum_{s_i \in \mathcal{S}^\star, WCD(s_i) > s_i.D} WCD(s_i) - s_i.D \\ \displaystyle\sum_{s_i \in \mathcal{S}^\star} WCD(s_i) - s_i.D \qquad \text{otherwise} \end{cases} \tag{2.3}$$

Where $WCD(s_i)$ is the WCD of a frame transmitted by a stream s_i having the UBS traffic class. We can then check if the frame is received by the deadline $s_i.D$. Such an analysis has been proposed in [SS16].

2.6.2 Heuristic Strategy

The Topology and Routing Heuristic (TRH) is a strategy which takes as input the set of end systems $\mathcal{ES}$, the components library $\mathcal{L} = (\mathcal{BT}, \mathcal{LT}, BC)$ and the set of streams $\mathcal{S}$, and returns the network topology $\mathcal{G}$ and the routing SR, see Algorithm 2.1.

TRH starts from a fully connected initial solution $\mathcal{G}_{init}$, line 1 in Algorithm 2.1 (similar to the solution discussed in 2.4a) onto which each stream in $\mathcal{S}^\star$ is routed, fulfilling the fault-tolerance requirements (the for loop in lines 3 to 11). Then, we remove from $\mathcal{G}_{init}$ the physical links and the bridges which are not used by the routing (line 12). TRH is a heuristic that does not guarantee finding the optimal solution, and it may terminate without finding a solution, even if one exists. The function *SearchRoute* returns a route for $mt_s(s_i)$ for a stream

Algorithm 2.1 $TRH(\mathcal{ES}, \mathcal{L}, \mathcal{S})$

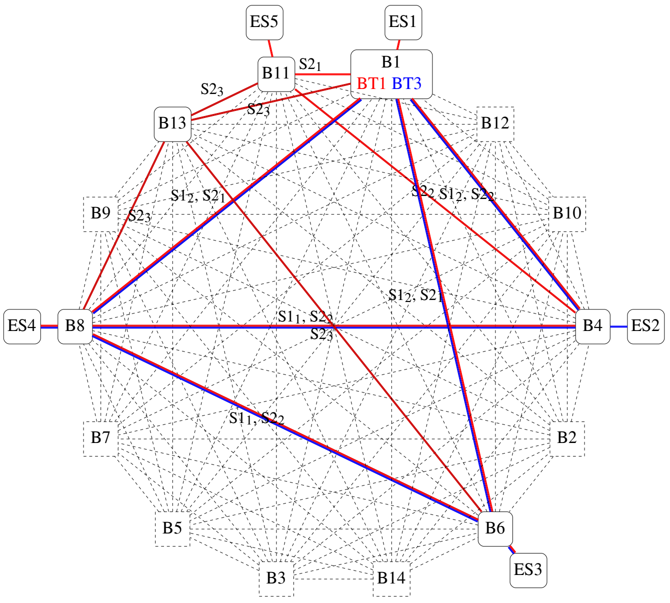
```
1: $\mathcal{G}_{init} \leftarrow CreateInitialTopology()$
2: $\mathcal{E}_{used} \leftarrow \emptyset$; $\mathcal{MT}_s \leftarrow \emptyset$
3: for $s_i \in \mathcal{S}$ do
4:    $\mathcal{G}_{ft} \leftarrow ConvertGraph(\mathcal{G}_{init})$
5:    for $j \in 1 : s_i.rl$ do
6:       $SR(s_i^j) \leftarrow SearchRoute(\mathcal{G}_{ft}, \mathcal{E}_{used}, s_i^j, s_i.rl - j)$
7:       $\mathcal{G}_{ft} \leftarrow RemoveEdges(\mathcal{G}_{ft}, Edges(SR(s_i^j)))$
8:       $\mathcal{E}_{used} \leftarrow \mathcal{E}_{used} \cup Edges(SR(s_i^j))$
9:       $\mathcal{G}_{init} \leftarrow AssignBridgeTypes(\mathcal{G}_{init}, SR(s_i^j))$
10:   end for
11: end for
12: $\mathcal{G} \leftarrow RemoveUnusedEdgesAndVertices(\mathcal{G}_{init}, \mathcal{E}_{used})$
13: $CheckSchedulability(\mathcal{S}, \mathcal{MT}_s)$
14: return $(\mathcal{G}, cost_A(\mathcal{G}), \mathcal{MT}_s)$
```

Id	Cost	No. int. ports	No. ext. ports
bt_1	8	2	3
bt_2	10	1	4
bt_3	16	2	5

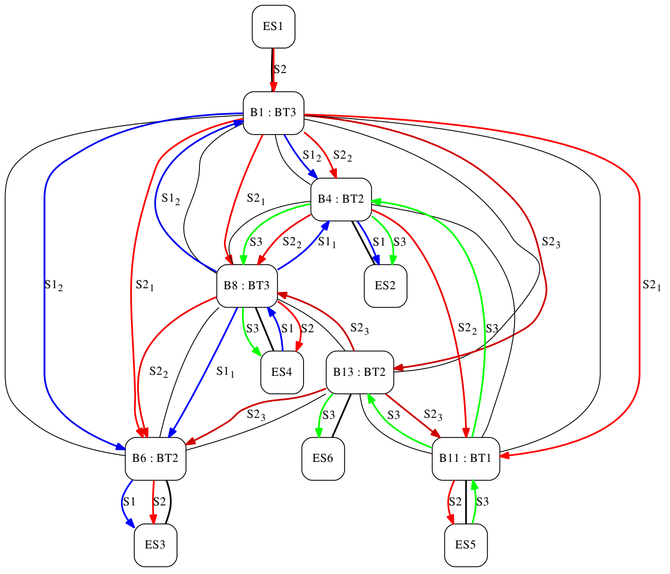
Table 2.3: Library for the TRH example

s_i. The idea of our heuristic is to keep track of the physical links used by the routes in $\mathcal{E}_{used}$, found for the already visited streams in Algorithm 2.1. $\mathcal{E}_{used}$ encourage subsequent calls to *SearchRoute* to reuse already used physical links as long as the fault-tolerance constraints are satisfied. TRH does not directly attempt to reduce the architecture cost $cost_A$ during this process, and it does not check the schedulability. Schedulability is checked in line 13 and $cost_A$ is reported for the constructed solution at the end.

$\mathcal{G}_{init}$ is obtained as explained in subsection 2.5.1 for 2.4a. Note that when assuming the bridge type for each bridge such that it accommodates the required ports, we also use bridge chaining if necessary, assigning the lowest-cost chain we can construct (see section 2.3 and Figure 2.2). Let us consider an example input with 6 ESes and the library $\mathcal{L}$ from Table 2.3, with the same link types presented in Table 2.1. Let us assume that the ESes ES_1 and ES_4 can be connected only to bridges of types bt_1 and bt_3, for ES_2, ES_3 and ES_6 are available bt_2 and bt_3 and ES_5 can be connected to bridge of type ES_1. Then, the $\mathcal{G}_{init}$ for this step is presented in 2.5a.



(a) Partial solution of TRH ($\mathcal{G}_{init}$ depicted with gray)



(b) Final solution of TRH

Figure 2.5: TRH example

TRH iterates through $s_i \in \mathcal{S}$ and determines the routes, lines 3-11 in Algorithm 2.1. We sort the streams in $\mathcal{S}$ based on the timing properties and RL (the aim is to route the most critical streams first). Moreover, when searching for a route for a redundant copy of a stream s_i^j we want to make sure that this route is disjoint to all the redundant routes established for s_i. Hence, we use the $\mathcal{G}_{ft}$ graph to keep track of already used links, removing from $\mathcal{G}_{ft}$ the edges involved in each $mt_s(s_i)$ determined so far (line 7). Thus, these edges will not be used in subsequent redundant routes for s_i. The route for s_i is searched on $\mathcal{G}_{ft}$, which for each original route $s_i \in \mathcal{S}$ considered is initialized to $\mathcal{G}_{init}$, where each undirected physical link is converted to two directed data flow links.

The routes are found using the *SearchRoute* function, which is an adapted Breadth-First Search (BFS) algorithm, presented in Algorithm 2.2. This function attempts to reuse as much as possible the edges used by previously determined routes, hence we keep track of the edges used so far in $\mathcal{E}_{used}$. After we determine a route, we update the bridge type for bridges in $\mathcal{G}_{init}$ by selecting from the library the bridge type of minimum cost which has the required number of ports. 2.5a shows a partial solution where we iterated over streams s_1 and s_2 and than determined the links and the stream's routes. The final step of TRH is to remove from $\mathcal{G}_{init}$ the edges not used for routes and to remove any vertices that became thus isolated in the topology (unused bridges), line 12 in Algorithm 2.1.

We modified BFS in *SearchRoute* function such that we visit a dataflow link during search only if *IsVisitable* returns true. *IsVisitable* returns true, if at least one of the following conditions holds: (1) the link was already used for the already determined routes or (2) the source and target bridges have enough free ports to support the addition of this link and of the next possible redundant copies. *IsVisitable* will return false if the dataflow link will exceed its capacity by routing s_i^j. In Algorithm 2.2 the function *Target* applied on a dataflow link gives the vertex on which the link enters and the elements of queue q are dataflow paths, therefore we used the function *LastVertex* to retrieve end of a dataflow path. The number of ports for a bridge is determined by summing up the number of physical links incident to that bridge that were used for already determined routings (stored in $\mathcal{E}_{used}$) and that are used for the current routing. If the stream for which we are searching the route is not the last one from the set of its redundant copies to the number determined before is added the number of remaining redundant copies, rl' in Algorithm 2.2.

Let us consider that for the TRH example considered earlier we have three streams, s_1 to s_3. Furthermore, let us assume that s_1 is sent from ES_4 to ES_2 and ES_3, s_2 from ES_1 to ES_3, ES_4 and ES_5 and s_3 is sent from ES_5 to ES_2, ES_4 and ES_6. For this example we consider that only first two streams are fault-tolerant, with a redundancy level of 2 and 3 for s_1 and s_2, respectively. 2.5b

Algorithm 2.2 *SearchRoute*$(\mathcal{G}_{ft}, \mathcal{E}_{used}, s_i^j, rl)$

```
1: $q \leftarrow \{s_i^j.src\}$
2: $dests \leftarrow s_i^j.dests$
3: $visited \leftarrow \emptyset; \mathcal{G}_{current} \leftarrow \emptyset$
4: $paths \leftarrow \emptyset$
5: while $q \neq \emptyset$ and $dests \neq \emptyset$ do
6:   choose first element of q as $current$
7:   $successors \leftarrow Successors(\mathcal{G}_{ft}, LastVertex(current))$
8:   for $succ \in successors$ do
9:     if $Target(succ) \notin visited$ and $IsVisitable(succ, \mathcal{E}_{used}, \mathcal{G}_{current}, rl, \mathcal{G}_{ft})$
       then
10:      $newPath \leftarrow current + succ$
11:      if $Target(succ) \notin \mathcal{ES}$ then
12:        $q \leftarrow q \cup \{newPath\}$
13:      else if $Target(succ) \in dests$ then
14:        $dests \leftarrow dests \setminus \{Target(succ)\}$
15:        $paths \leftarrow paths \cup \{newPath\}$
16:        $\mathcal{G}_{current} \leftarrow \mathcal{G}_{current} \cup Edges(newPath)$
17:      end if
18:    end if
19:  end for
20:  $visited \leftarrow visited \cup \{Target(succ)\}$
21: end while
22: return $ConvertToTree(paths)$
```

shows the final solution of TRH for our example, i.e., the routes and the network from which the unused physical links and bridges are removed. The physical links are depicted with solid black lines (the internal links use thicker lines) and the routes are depicted with colored thin arrows. We used blue arrows for routes of s_1, red for s_2 and green for the route of s_3. The networks in Figure 2.5 have been generated by our tool, and redundant streams s_i^j are labeled as Si$_\text{j}$.

2.6.3 GRASP

GRASP [FR89] is a meta-heuristic optimization, which searches for that solution which minimizes the $cost_T$ function. GRASP is an iterative algorithm, where each iteration consists of two phases: Phase (i) constructs an initial solution (a topology and a route for each stream $s_i \in \mathcal{S}^\star$) based on a randomized greedy algorithm and Phase (ii) performs a local search on the constructed solution to reach the local minimum. At the end of each iteration, if the cost of the local minimum found is less than the cost of the best solution, found so far, the solution is stored as the "best-so-far". The termination condition for the strategy is based on a given time limit. We implemented GRASP with Google OR-Tools.

To construct the solutions in Phase (i), we have adapted our TRH strategy as follows. First, we create initial solutions by creating a random ordering of streams at the start of Algorithm 2.1 (with TRH, the streams are ordered based on their "criticality" of timing and RL). We use the same *SearchRoute* function (Algorithm 2.2). Then, we also create initial solutions which do not use the routes returned by *SearchRoute*, but instead use random routes, in the hope of providing a better coverage of the search space.

In Phase (ii), starting from each such initial solution, we search for a local minimum using the Large Neighborhood Search (LNS) algorithm [Sha98], which improves the initial solutions by iteratively "destroying" and "repairing" the solution.

For the destroy part we use 6 operators which remove links from the topology (removing all routes routed over that links) or remove routes. The operators are as follows: (1) remove a link, (2) remove two routes, (3) remove a route with the containing links and routes routed over these links, (4) select one stream $s_i \in \mathcal{S}^\star$ and remove the routes for its original stream and redundant copies, (5) select two streams $s_i, s_j \in \mathcal{S}^\star$ and remove the routes for their original streams and redundant copies and (6) select two original streams $s_i, s_j \in \mathcal{S}$ and for each stream remove the route for a randomly chosen redundant copy. The *degree of destruction* is such that we are able to explore the neighborhood in a reasonable time, but we do not degrade into a full optimization.

Several options are possible for the "repair". We have decided to use an "optimal repair", i.e., the best possible full solution is constructed from the partial solution. To find the solution for the repair we have used the CP-Strategy presented in the next section.

2.6.4 Constraint Programming-Based Strategy

The Topology and Routing Optimization (TRO) strategy is a Constraint Programming (CP) [Apt03] model. TRO takes as input the set $\mathcal{ES}$, the components library $\mathcal{L}$ and the set $\mathcal{S}$, and attempts to determine the optimal network architecture $\mathcal{G}$ and the routing SR according to the cost function introduced in subsection 2.6.1. Moreover, it can take $\mathcal{G}$ as an additional input and attempts to only determine the optimal routing SR for the given architecture. In the following, we present a CP model for the problem described in section 2.5.

2.6.4.1 CP Model

We use two parameters: (1) $n = |\mathcal{ES}|$ the number of End Systems. (2) $n_{max} =$ the maximum number of bridges, which can be given or computed as explained in subsection 2.6.2 for 2.5a. Based on these parameters we define the following sets:

- K End Systems index set, $K = \{1, .., n\}$
- J bridges index set, $J = \{n+1, n+2, .., n+n_{max}\}$
- I End Systems and bridges index set, $I = K \cup J$

To model the topology of the network let $\mathcal{A}_{i1,i2}$ denote the adjacency matrix of the $\mathcal{G}_{max}(\mathcal{V}_{\max}, \mathcal{E}_{\max})$ which has $n + n_{max}$ vertices. Each element of this matrix is an integer variable that represents the type of the link between nodes of this graph, $\forall i1, i2 \in I \; a_{i1,i2} \in \{0, .., |\mathcal{LT}|\}, lt_{a_{i1,i2}} \in \mathcal{LT} \cup \{0\}$ ($0 =$ no link). Since the topology graph is undirected, we only initialize new variables for the right-upper half of the matrix elements. The elements on the other half of the matrix point to their symmetric elements, therefore $\forall i1, i2 \in I \; a_{i1,i2} = a_{i2,i1}$. Since the graph should not contain self-loop links, we set the elements on the main diagonal of the matrix to 0, $\forall i \in I \; a_{i,i} = 0$. We also define $\mathcal{BA}_j$ which is an integer vector to specify the type of the bridges in the network. For each bridge $j \in J$, variable ba_j specifies the type of the bridge ($ba_j \in \{1, .., |\mathcal{BT}|\} \cup \{nil\}, bt_{ba_j} \in \mathcal{BT} \cup \{nil\}$). If this value is *nil* for a bridge j, it means that the bridge is not active and it is not included in the network topology.

In order to model the stream's routes, we define two integer matrices $\mathcal{MT}^{s}_{s^\star,i}$ and $\mathcal{MT}^{w}_{s^\star,i}$. Both matrices have the same size and dimensions. The size of the first dimension is the size of all the streams including their redundant copies ($|\mathcal{S}^\star|$), and the size of the second dimension is the size of all the vertices including End Systems and bridges ($|I|$). For each stream (including the redundant copies $s \in \mathcal{S}^\star$), $\mathcal{MT}^{s}_{s}$ is an integer vector that specifies (in backwards order) a multicast tree for the stream. Each element of this vector ($\forall s \in \mathcal{S}^\star, i \in I \; \mathcal{MT}^{s}_{s,i} \in I \cup \{nil\}$), holds the successor vertex on the path from the vertex i to the *source* of the stream s. In the same manner, each element of $\mathcal{MT}^{w}_{s,i}$ holds the weight of the path from the vertex i to the *source* of the stream s, $\forall s \in \mathcal{S}^\star, i \in I \; \mathcal{MT}^{w}_{s,i} \in \mathbb{R}$. The value of $\mathcal{MT}^{s}_{s}$ for the source of the stream is set to the index of the source End System, $\mathcal{MT}^{s}_{s,s.src} = s.src$ and $\mathcal{MT}^{w}_{s,s.src} = 0$. If a bridge is not part of the multicast tree then $\mathcal{MT}^s_{s,i} = nil$ and $\mathcal{MT}^w_{s,i} = -1$. Our mathematical model and constraints for multicast tree are an extension of the models presented in [PD12].

According to these variables, we define the following constraints.

2.6.4.2 Topology Constraints

1. Any End System, $ES \in \mathcal{ES}$, must be connected with one bridge: $\forall k \in K \ \sum_{j \in J}(a_{k,j} \neq 0) = 1$
2. The specified bridge constraint BC should be satisfied: $\forall k \in K, j \in J \ a_{k,j} \neq 0 \Rightarrow (bt_{ba_j}, lt_{a_{k,j}}) \in BC(ES_k)$
3. The number of internal links connected to a bridge should not exceed the number of internal ports supported by the bridge:
   $\forall j \in J \ \sum_{i \in I, a_{i,j} \neq 0}(lt_{a_{i,j}}.lct = Int) \leq bt_{ba_j}.noIntPorts$
4. The number of external links connected to a bridge should not exceed the number of external ports supported by the bridge:
   $\forall j \in J \ \sum_{i \in I, a_{i,j} \neq 0}(lt_{a_{i,j}}.lct = Ext) \leq bt_{ba_j}.noExtPorts$
5. If a bridge is inactive, it cannot be connected to other bridges or end systems: $\forall j \in J \ \sum_{i \in I} a_{i,j} = 0 \Leftrightarrow ba_j = nil$
6. All the active bridges should be connected at least via two links (no bridge is an end point): $\forall j \in J \ \sum_{i \in I, i \neq j}(0 < a_{i,j}) \geq 2 \times (ba_j \neq nil)$

2.6.4.3 Routing Constraints

1. The successors of destinations of a stream cannot be *nil*:
   $\forall s \in \mathcal{S}, \forall d \in s.dests \ \mathcal{MT}^s_{s,d} \neq nil$
2. The successors of nodes which are not the source should not point to themselves: $\forall i \in I, s \in \mathcal{S}^\star, i \neq s.src \ \mathcal{MT}^s_{s,i} \neq i$
3. The successors of End Systems which are neither source nor destinations of a stream must be *nil*: $\forall s \in \mathcal{S}, k \in K \backslash \{s.src\} \backslash s.dests \ \mathcal{MT}^s_{s,k} = nil$
4. Inactive bridges cannot be used within a multicast tree:
   $\forall j \in J \ \sum_{s \in \mathcal{S}^\star} \left(\mathcal{MT}^s_{s,j} \neq nil\right) \neq 0 \Leftrightarrow ba_j \neq nil$
5. The multicast trees must not contain cycles: $\forall s \in \mathcal{S}^\star, i \in I \backslash \{s.src\}$
   $\mathcal{MT}^s_{s,i} \neq nil \Rightarrow \mathcal{MT}^w_{s,i} = \mathcal{MT}^w_{s,\mathcal{MT}^s_{s,i}} + 1$

6. All the bridges should be transient, which means that if a stream enters to a bridge through a link it should exit from another link connected to the bridge: $\forall s \in \mathcal{S}^{\star}, i \in I, \exists j \in J, i \neq j$ s.t.$\mathcal{MT}^{s}_{s,i} \neq nil \Leftrightarrow \mathcal{MT}^{s}_{s,j} = i$

7. Any vertex and its successor should be connected: $\forall s \in \mathcal{S}^{\star}, i1, i2 \in I\ \mathcal{MT}^{s}_{s,i1} = i2 \Rightarrow a_{i1,i2} \neq 0$

8. All the multicast trees for the redundant copies of each stream should not have common links: $\forall s \in \mathcal{S}, v \in I \backslash \{s.src\} \backslash s.dests, i, j = 1, \ldots, s.rl\ i \neq j \wedge i \neq nil \wedge j \neq nil \Rightarrow \mathcal{MT}^{s}_{s^{i},v} \neq \mathcal{MT}^{s}_{s^{j},v}$

9. For all physical links, the capacity of the links should not be exceeded: $\forall i1, i2 \in I\ \sum_{s \in \mathcal{S}^{\star}} \big((\mathcal{MT}^{s}_{s,i2} = i1) \times s.R \big) \leq lt_{a_{i1,i2}}.speed$

2.6.4.4 Search Strategy

We define $\mathcal{MT}^{s}_{s^{\star},i}$ and $\mathcal{BA}_j$ as the main decision variables. Consequently, the assignments of $\mathcal{A}_{i,i}$ and $\mathcal{MT}^{w}_{s^{\star},i}$ will be determined by propagating the constraints (8) and (6). In the case that the architecture is given as input, we initialize the values of $\mathcal{A}_{i,i}$ and $\mathcal{BA}_j$ according to the given architecture. Thus, the solver will do the exhaustive search only for $\mathcal{MT}^{s}_{s^{\star},i}$.

Two strategies should be specified for the CP solver to perform the search. The first is the order of selecting the variables for assignment. The other strategy is the order of selecting the values from the variable's domain for assignment. Based on the results obtained for small case studies, we decide to use First-Unbound-Variable and Assign-Min-Value strategies for the decision variables.

To validate the schedulability constraint and guide the solver to find the optimal solution, we implemented a Search-Monitor (the term used in OR-Tools) that will be triggered whenever the CP solver finds a solution (which satisfies all the constraints). The search-monitor computes the degree of schedulability δ_t and the architecture cost $cost_A$ of the obtained solution. If the degree of schedulability is less or equal to zero, it will consider the solution as a feasible solution, and if the total cost $cost_T$ of the solution is less than the cost of the earlier solutions, it will consider the solution as the best solution obtained so far. At the end of the search process, we will return the best solution found by converting the assignments of $\mathcal{A}_{i1,i2}$ and $\mathcal{BA}_j$ into the network architecture $\mathcal{G}$, and $\mathcal{MT}^{s}_{s^{\star},i}$ into the routing SR obtained for the given set of streams.

2.7 Experimental Results

For the evaluation of our strategies, namely the Topology and Routing Heuristic (TRH), which is a heuristic approach, GRASP (a metaheuristic strategy), and the Topology and Routing Optimization (TRO), which is a CP-based strategy, we used five synthetic test-cases, *motiv*, *tc1* to *tc4* and *GM*, which is a real-life case study from General Motors. For all experiments, as the components library for the link types we used those presented in Table 2.1, but for bridges, we have extended the library to contain 6 bridge types. For these experiments we considered that all streams have the same priority. In Table 2.4 the first 3 columns describe the test-case, i.e., name, number of end systems, and the number of streams and their redundant copies. The strategies, TRH, GRASP and TRO were implemented in Java and all experiments were run on Intel Core i7-2600 machines at 3.4 GHz. For TRO and GRASP we have used OR-Tools [vOPF16], which is a CP library introduced by Google.

The results of our experiments are shown in Table 2.4. Column 4 shows the cost of the fully connected topology $\mathcal{G}_{init}$ (see subsection 2.6.2), which is an upper bound on the architecture cost. For all strategies, we present the architecture cost and the execution time. These strategies were able to find schedulable solutions, for all test-cases.

As it can be observed in Table 2.4, our strategies are able to significantly reduce the architecture cost $cost_A$. Compared to $\mathcal{G}_{init}$ TRH is able to decrease the architecture cost, in average with 37%, with a maximum decrease in cost for *GM*, where the cost is reduced by 80%. Although TRH is able to decrease architecture cost compared to $\mathcal{G}_{init}$ and scales well with the problem size, it obtains solutions with a higher cost compared to GRASP and TRO. For example, TRO improves on average by 27% on the TRH architecture cost, and GRASP improves on average by 23% compared to TRH.

TRO is able to find the optimum solution, marked by * in the table. However, TRO finds the optimum solution only for the smaller test-cases *motiv*, *tc1* and *tc2*, and it does not scale well with the problem size. For the other test-cases, TRO did not find the optimum solution and we list in the table the architecture cost found after a time limit (execution time) of 48 hours that we impose on the search.

Finally, our proposed GRASP strategy is able to obtain good quality results in a reasonable time. As we can see, GRASP obtains optimum solution for *motiv* test-case, and for the other test-cases, it obtains results comparatively close to the TRO with the relative gap of 5% on average. The main advantage of GRASP is that it can explore the design space much faster. For example,

Name	$\|\mathcal{ES}\|$	$\|\mathcal{S}\|$, $\|\mathcal{S}^\star\|$	$\mathcal{G}_{init}$	*TRH*		*GRASP*		*TRO*	
			$cost_A$	$cost_A$	Exec. time (s)	$cost_A$	Exec. time (s)	$cost_A$	Exec. time (s)
motiv	4	3, 4	78	63	0.15	*43	0.67	*43	1.32
tc1	4	4, 6	78	78	0.09	50	0.40	*41	11.84
tc2	4	8, 11	78	71	0.08	52	0.60	*41	32.6
tc3	6	6, 8	176	106	0.1	76	0.95	73	48 h
tc4	15	30, 34	1893	392	8.05	336	3.35 min	357	48 h
GM	20	27, 38	2230	432	130	402	9.3 min	410	48 h

Table 2.4: Experimental results

for the realistic test case GM, GRASP has obtained an architecture cost of 402 in 9.3 minutes, compared to TRO, which has actually obtained a larger cost of 410 in the 48 hours we let it run.

2.8 Conclusions

In this paper, we have considered safety-critical real-time applications implemented using TSN-based distributed architectures. Our focus was on the synthesis of the network topology and streams routing such that the real-time and redundancy requirements of the applications are satisfied, and the cost of the architecture is minimized. We have proposed three strategies, a heuristic approach, called Topology and Routing Heuristic, a GRASP metaheuristic and an approach based on CP, namely Topology and Routing Optimization. The experimental results show that by using our strategies we are able to significantly reduce the cost of architecture, obtaining architectures which are at the same time fault-tolerant and meet the timing requirements of the streams. In particular, the proposed GRASP metaheuristic is able to obtain good quality results in a reasonable time, and scales well with the problem size.

Chapter 3

Paper B: Traffic Type Assignment for TSN-based Mixed-Criticality Cyber-Physical Systems

In this paper we are interested in mixed-criticality applications, which have functions with different timing requirements, i.e., hard real-time (HRT), soft real-time (SRT) and functions that are not time-critical (NC). The applications are implemented on distributed cyber-physical systems that use IEEE Time-Sensitive Networking (TSN). TSN is an IEEE effort to bring deterministic real-time capabilities to IEEE 802.3 Ethernet. TSN supports the convergence of multiple traffic types, i.e., critical, real-time and regular "best-effort" traffic within a single network: Time-Triggered (TT), where messages are transmitted based on static schedule tables; Audio-Video Bridging (AVB), for dynamically scheduled messages with a guaranteed bandwidth and bounded delays; and Best Effort (BE), for which no timing guarantees are provided. HRT messages have deadlines, whereas for SRT messages we capture the quality-of-service using "utility functions". Given the network topology, the set of application messages, including their routing, and the set of available AVB classes we are interested to determine the traffic type of each message, such that all HRT messages are

schedulable and the total utility for SRT messages is maximized. We propose a Tabu Search-based metaheuristic to solve this optimization problem. The proposed approach has been evaluated using several benchmarks, including two realistic test cases.

3.1 Introduction

Mixed-criticality cyber-physical systems have functions with different safety-criticality requirements, e.g., highly critical, mission critical, non-critical. For example, a network backbone in a modern vehicle has to integrate Advanced Driver Assistance Systems (ADASes) functions, which rely on high-bandwidth data from sensors, e.g., video cameras and LIght Detection And Ranging (LIDAR), with powertrain functions that have tight timing constraints but use small frame sizes, and diagnostic services, which are not time-critical. Due to the increase in complexity, and the need to reduce costs, such mixed-criticality applications are today implemented in *integrated architectures*, where functions of different criticality share the same distributed platform.

There are several communication protocols on the market, depending on the application area, e.g., FlexRay for automotive, ARINC 664 p7 for avionics [Aer09], and EtherCAT for industrial automation [JB04]. However, emerging applications, e.g., ADAS, autonomous driving, or Industry 4.0, have increasing bandwidth demands. For instance, autonomous driving requires data rates of at least 100 Mbps for graphical computing based on camera, radar, and LIDAR data, whereas CAN and FlexRay only provide data rates of up to 1 Mbps and 10 Mbps, respectively. In addition, although there have been many safety-critical protocols proposed, only few of them can support the separation required by mixed-criticality messages [Rus01].

TSN: The well-known networking standard IEEE 802.3 Ethernet [IEE12] meets the emerging bandwidth requirements for safety-critical networks, while remaining scalable and cost-effective. However, Ethernet is known to be unsuitable for real-time and safety-critical applications [Dec05]. In Ethernet networks, messages are transmitted between end systems as *frames*. Frames are forwarded on links, through switches, on a route from a sender to one or multiple receivers. They queue up in switches during transmission while waiting for the next link in the route to become available. Each switch has multiple queues, and frames are filtered into queues based on their priority. When a link becomes available a new frame is chosen for transmission starting from the highest priority queue. Hence, the queueing delay for each frame depends on its priority, on how many other frames are queued in front of it, and on the availability of the next link.

This leads to network congestion causing nondeterministic behavior.

Many extensions, such as EtherCAT [JB04], PROFINET [Fel04], ARINC 664p7 [Aer09], and TTEthernet [SAE11], have been suggested and are used in the industry. Although they satisfy the timing requirements, they are incompatible with each other, and as a result, they cannot operate on the same physical links in a network without losing real-time guarantees [DN16]. Consequently, the IEEE 802.1 Time Sensitive Networking task group [TSN12] has been working since 2012 on standardizing real-time and safety-critical enhancements for Ethernet. TSN consists to a large extent of amendments (we will call them “sub-standards”) to IEEE 802.1Q[1]. TSN supports multiple traffic types and is thus suitable for mixed-criticality applications: Time-Triggered (TT) traffic for applications with tight timing constraints, Audio-Video Bridging (AVB) for applications that need bounded latency but may have less stringent timing requirements, and Best-Effort (BE) traffic for non-critical applications that do not need timing guarantees. Therefore, in this paper we are interested in cyber-physical systems which use IEEE 802.1 Time-Sensitive Networking (TSN) for communication.

AVB: First, the IEEE 802.1Q-2005 sub-standard introduced support for prioritizing the BE traffic in order to improve its Quality of Service (QoS). Following this, the IEEE Audio-Video Bridging (AVB) Task Group was formed to develop another set of enhancements, namely IEEE 802.1BA Audio Video Bridging Systems, known as AVB. AVB frames are compliant with the sub-standard 802.1BA. AVB messages may be delayed by other AVB messages or by TT traffic, which has the highest priority. However, analysis methods exist that bound their Worst-Case end-to-end Delays (WCDs) [ZPZL18], providing thus timing guarantees. AVB uses the Credit-Based Shaper (CBS) from IEEE 802.1BA to prevent the starvation of lower priority flows such as BE. In 2012, the AVB Task Group was renamed to IEEE 802.1 Time-Sensitive Networking Task Group to reflect the shifted focus onto further extending the protocol towards safety-critical and time-sensitive applications.

TT: The sub-standard IEEE 801.Qbv Enhancements for Scheduled Traffic introduces time-aware gates within network devices enabling fully deterministic temporal behavior of real-time communication. For each egress port of a switch, a schedule table called *Gate Control List* (GCL) specifies which queue may transmit at which points in time. Using this functionality, enables frames to be forwarded in the network in a time-triggered manner. A prerequisite for time-triggered communication is the presence of a network-wide reference time, such as, the IEEE 802.1AS synchronization protocol that allows local clocks in the end

[1]We will not provide references for all sub-standards, but these can be easily found based on their names via IEEE Xplore.

stations and switches to synchronize to each other. Thus, TT frames have the highest priority and are transmitted based on synchronized distributed schedule tables. By synthesizing carefully the GCLs, we can guarantee the schedulability of TT frames, ensuring also low end-to-end latency and low jitter [CSCS16].

BE: BE messages are compliant with IEEE 802.3 Ethernet and have the lowest priority, without any timing guarantees.

Problem formulation: In this paper we are interested in mixed-criticality applications, which have functions with different timing requirements, i.e., hard real-time (HRT), soft real-time (SRT) and functions that are not time-critical (NC). In our model, HRT messages have hard deadlines, whereas for SRT messages we capture the QoS using soft deadlines and “utility functions”, which model the relative importance of SRT messages and how the performance of the system degrades if the SRT soft deadlines are missed. Similar to the debate in real-time systems between time-triggered and event-triggered implementations [PPEP08][PSS15], there is no agreement on the appropriate traffic type for the mixed-criticality messages, which depends on the particularities of the applications. Therefore, in this paper, we are interested in the problem of *Traffic Type Assignment* for mixed-criticality messages in TSN.

Given the network topology, the set of application messages, including their routing, and the set of available AVB classes we are interested to determine the traffic type for each message, such that all HRT messages are schedulable and the total utility for SRT messages is maximized. For the TT frames we decide their TT queue assignment and schedule tables, and for the AVB traffic we decide its AVB class assignment and the corresponding parameters influencing its latency, i.e., idle slopes for CBS. We consider that the NC messages are implemented using the BE traffic type, and we do not consider the BE traffic type for HRT or SRT messages. However, the HRT and SRT messages can be implemented with the TT or the AVB traffic types, as both types provide real-time guarantees. In case the NC messages require QoS guarantees, they can be treated as SRT messages. We propose a Tabu Search-based metaheuristic to solve this optimization problem.

Contribution: We have discussed briefly the problem of Traffic Type Assignment for TTEthernet, which uses TT and Rate-Constrained (RC) traffic classes. The solution for TTEthernet is not applicable for TSN, which handles TT traffic differently, and which uses AVB that differs significantly from RC, see the “TSN vs TTEthernet” discussion in the next section. Note that in context of TSN we use the term “traffic type” to clearly differentiate it from the concept of AVB “classes”. To the best of our knowledge, this is the first time such a problem has been addressed in the context of TSN.

The paper is structured as follows. The next two sections present the architecture and application models, respectively. The TSN protocol is presented in section 3.4, where we introduce the details of the TT and AVB traffic types. The formulation of our problem and a motivational example are detailed in section 3.5. section 3.6 presents our Tabu Search-based metaheuristic solution, which is evaluated in section 3.7. The last section presents our conclusions and a discussion.

3.1.1 Related Work

In the context of real-time cyber-physical systems, there have been several comparisons between time-triggered (TT) and event-triggered (ET) approaches, both at the task-level [PPEP08], and at message-level [PSS15][Kop91]. In [PPEP08], the authors decide which tasks should be TT and which ET, showing that the right choice depends on the particularities of the applications. The ET and TT approaches are compared, in [Kop91], in terms of predictability, resource utilization and scalability. The advantages are that the TT-based communication is more predictable, while using an ET-based approach less resources and configuration is required. But, as the researchers show, the system scalability is reduced for both approaches; after addition of tasks, traffic or network components ET-based systems have to be retested for temporal constraints, but in contrast, for a TT-based architecture new schedule tables should be created and integrated. Researchers [PSS15] have also compared two networking approaches, i.e., Time-Division Multiplexing (TDM) with an ET approach in the context of Networks-on-Chip. Their conclusion, inferred experimentally, is similar with the theoretical presentation from [Kop91], namely that ET improves the bandwidth usage, whereas TDM is suited when the latencies have to be reduced.

In the context of ARINC 664 p7 [Aer09], which is a standard from the avionics area that defines the “Rate Constrained” (RC) ET traffic type, researchers have shown how to optimize the priorities of the RC traffic [HSF14]. They propose an extension to the Optimal Priority Assignment algorithm used for real-time tasks by assigning higher priorities to traffic with more stringent timing requirements. Another strategy of priority assignment, for CAN-based solutions, is presented in [NSS00], where messages’ priorities are selected based on reliability analysis.

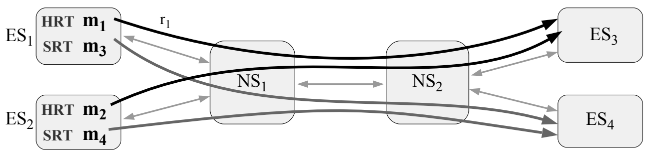
In this paper we are interested in “Deterministic Ethernet” protocols that can support multiple traffic types. Recent developments in TSN have also introduced a new traffic type called Urgency-Based Scheduler (UBS), see the IEEE 802.1Qcr sub-standard [TSN18b]. UBS is intended as a replacement for both TT and AVB traffic. Although a comparison between UBS and TT is interesting to investigate in future work, addressing UBS is beyond the scope of this paper. Researchers have shown how to decide the frame priorities at the switch level for UBS [SS16]. Besides TSN, introduced in the previous section (and presented in more detail in section 3.4), another Deterministic Ethernet protocol that supports multiple traffic types is TTEthernet. TTEthernet is a deterministic, synchronized and congestion-free communication protocol based on Ethernet, compliant with ARINC 664 p7 [Aer09] protocols and using the clocks synchronization strategy defined in SAE AS6802 [SAE11].

For both TSN and TTEthernet, researchers have proposed methods to synthesize the communication schedule tables, see [PSRH15][CS16] for TTEthernet and [PRCS16][CSCS16][SCS18] for TSN. The most scalable of these methods are able to handle up to 100,000 TT messages, but they ignore non-scheduled traffic such as RC in TTEthernet (TSN has not been considered). The problem of routing has been addressed for TSN for both the TT traffic type [Nay17][SDT$^+$17] [PTO18] (and thus integrated with scheduling) and for AVB [Lau16]. The findings show that the routing has an influence on the WCDs of both TT and AVB traffic. In this paper, to facilitate a fair comparison of TT and AVB in terms of traffic type, we consider that the routing is given and fixed, determined using methods such as [Nay17] for TT and [Lau16] for AVB.

TSN vs. TTEthernet: Although the TT traffic types in TSN and TTEthernet have similarities, they differ significantly on how they schedule the TT frames. TTEthernet schedule tables are specified for individual frames, whereas TSN specifies schedules for the queues (the GCLs), not frames. Consequently, all frames sharing the same queue are affected by the associated schedule table. Because TSN schedules queues, TT frames can interfere with each other. When a frame is scheduled for transmission on a link in a given time interval, the corresponding GCL is set to open the associated gate in that interval. Non-determinism can be introduced, for example, if frames incoming on different ingress ports arrive at roughly the same time or if a frame is lost due to faults. The non-determinism will compromise the timeliness of TT frames.

For this reason, researchers have proposed GCL synthesis solutions [CSCS16] [SCS18] that enforce determinism, such that only frames of one of the flows are present in the queue when the associated gate is open. This restricts the solution space for the GCLs. Recent work [ZPZL18] has relaxed these restrictions, and has presented a method to determine the WCDs of TT frames considering arbitrary GCLs, which are given as an input. However, in this work we consider that the TT frames are scheduled in a fully deterministic fashion, which allows better control of their latency and jitter via the GCL synthesis, without the need of a schedulability analysis for TT.

Both RC in TTEthernet and AVB in TSN are ET traffic classes. However,



(a) Example architecture model

Msg.	Size (in B)	Period (in ms)	Deadline / (Utility)
$m_1 \in \mathcal{M}^{HRT}$	50	2	1 ms
$m_2 \in \mathcal{M}^{HRT}$	62.5	3	2 ms
$m_3 \in \mathcal{M}^{SRT}$	200	4	1.5 ms / (max. 6; 0 at 2.6 ms)
$m_4 \in \mathcal{M}^{SRT}$	270	4	2.5 ms / (max. 6; 0 at 4.1 ms)

(b) Example application model

Figure 3.1: Example system model

RC uses the concept of "Virtual Links" from ARINC 664 p7 [Aer09], which are parameterized via a Bandwidth Allocation Gap (BAG), the minimum time interval between two consecutive instances of an RC frame, to ensure bounded delays. AVB in TSN uses the Credit-Based Shaper (CBS) from IEEE 802.1BA, configured via "slopes", i.e., an idle slope and a sending slope.

Although a lot of work is being done for the analysis and optimization of Deterministic Ethernet-based cyber-physical systems [PRCS16], all of the related work has assumed that the traffic types are manually decided by the system engineer. The problem of Traffic Type Assignment has been addressed briefly only for TTEthernet in [GP16].

3.2 Architecture Model

We model a TSN network as an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where the vertices (or nodes) $\mathcal{V} = \mathcal{ES} \cup \mathcal{NS}$ denote the set of all End Systems (ESes) and Network Switches (NSes, also called bridges), respectively, and the edges $\mathcal{E}$ are the physical links interconnecting the ESes and NSes. Without loss of generality, we assume that all physical links have the same transmission rate C. An ES is composed of a CPU, memory, I/Os, and a network interface card. TSN provides the clock synchronization needed for the schedule tables of the TT type. 3.1a shows an example TSN-based CPS with 4 ESes and 2 NSes.

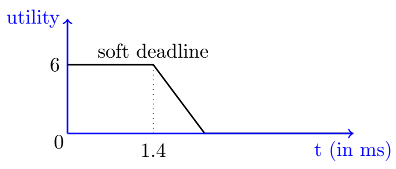
A dataflow link (DL) dl_i represents a directed connection between two nodes in $\mathcal{V}$. A sender task in a source ES is connected to a receiver task in a destination ES through a dataflow path (DP) dp_i, which is a sequence of interconnected DLs. The set of all DLs is denoted with $\mathcal{DL}$ an the set of all DPs is $\mathcal{DP}$. A frame in TSN has one source, but it may have multiple receivers. The route for each message is given and fixed for all frames. We model a route r_i as a set of DPs, one for each receiver, and all DPs sharing the same sender. $\mathcal{R}$ is the set of all routes. 3.1a shows 4 routes. For example, the frame of message m_1 from ES_1 to ES_3, has the routing r_1 consisting of ES_1, NS_1, NS_2, ES_3 as depicted with a thick black arrow.

3.3 Application Model

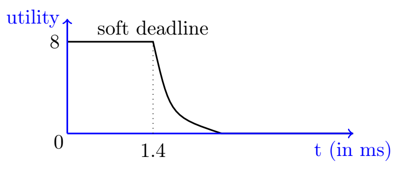
In this paper, we consider mixed-criticality applications. Our application model captures the messages in the applications, and their timing requirements. We consider three types of timing criticality: Hard-Real Time (HRT) messages, which have strict deadlines, Soft Real-Time (SRT) messages, for which we are interested to maximize their "utility" and Non-Critical (NC) messages, which have no timing requirements. We denote the set of all messages in a cluster with $\mathcal{M} = \mathcal{M}^{HRT} \cup \mathcal{M}^{SRT} \cup \mathcal{M}^{NC}$, where the three sets correspond to the set of all HRT, SRT and NC messages.

Each message $m_i \in \mathcal{M}$ has a source ES_i^{src} and one or more destinations $\mathcal{ES}_i^{dest}$, and a given size $m_i.size$. HRT and SRT messages are periodic, with a period $m_i.period$. Both HRT and SRT messages have a deadline, $m_i.deadline$. The HRT deadline is *hard*, i.e., if the deadline is missed, it may result in catastrophic consequences. The SRT deadline is *soft*, i.e., the performance of the system degrades if the deadline is missed. A single Ethernet frame transmits a payload of at most 1,500 bytes (B), the so-called Maximum Transmission Unit (MTU). If the data size is larger than MTU, the message is fragmented into multiple frames. Thus, a message m_i will be split into $k = \lceil \frac{m_i.size}{MTU} \rceil$ frames $f_{i,1}, f_{i,2} \ldots f_{i,k}$.

For SRT messages we use a positive non-increasing utility function, denoted with $m_i.utility(t)$, where t is a time instant relative to start of the message transmission from its source ES_i^{src}. The utility starts from a positive value and sometimes after the soft deadline reaches a zero value. We consider that the system engineer specifies the utility functions of SRT messages to capture their relative importance and how the performance of the system degrades if their soft deadlines are missed, see [BLAC05] for a discussion on utility functions. If a SRT message m_i arrives within its soft deadline $m_i.deadline$, then its utility value is maximal. However, if the deadline is missed, the utility value will decrease with



(a) Example Linear Utility Function



(b) Example Hyperbolic Utility Function

Figure 3.2: Examples Utility Functions

time, as specified by the definition of the utility function $m_i.utility(t)$.

Let us explain now, using the example from Figure 3.2, how nonincreasing positive utility functions capture how the service degrades by time when the message is received after its soft deadline and the SRT messages relative importance. The function starts as a constant function from a positive value, 6 in 3.2a and 8 in 3.2b. The utility value starts to decrease linearly 3.2a and hyperbolically 3.2b after the soft deadline, 1.4 ms in both examples reaching a value of zero. The maximal utility value of 8 indicates that the soft real-time message from 3.2b is more important than the message from 3.2a, which has a smaller maximum value.

3.1b shows an example[2] application model, with two HRT messages m_1 and m_2 and two SRT, m_3 and m_4. Their routing is presented in 3.1a. In this example, we use a simple linear utility function for both SRT messages m_3 and m_4, starting at a maximum utility of 6, linearly decreasing after the soft deadline, reaching a utility of zero at 2.6 ms and 4.1 ms, respectively. Our model does not explicitly capture NC messages, which we assume that will always be assigned to the BE traffic type.

3.4 TSN Protocol

We present in this section how the traffic types in TSN are being transmitted. The presentation is not intended to be exhaustive; instead, it focuses on the concepts needed in this paper. For details, the reader is directed to the standards mentioned in the text.

Each egress port in a bridge has eight queues associated with it, and each

[2]Ethernet frames sizes are constrained between 64 B and 1518 B (including overhead), but in this illustrative example we use smaller values for simplicity.

queue has a priority, from seven (highest) to zero (lowest), see Figure 3.3 for an illustration. Every frame contains a priority field determining which queue to be placed in. TT traffic can use one or more queues, and these queues have the highest priority. The number of TT queues N_{TT} and the assignment of TT frames to TT queues $Q_{TT}(m_i)$ is determined by our optimization approach. The remaining queues are used by AVB or BE traffic. The AVB queues make use of a Transmission Selection Algorithm (TSA) in the form of the *Credit-Based Shaper* (CBS), explained in subsection 3.4.2. The transmission of TT frames is explained in the next section, but BE frames will not be further covered.

The *Transmission Selection* (see Figure 3.3) initiates transmission from the highest priority queue that is *available* and has frames to transmit. The availability of each queue is controlled by (i) its transmission *Gate*, which can either be in an *open* or *closed* state and (ii) a CBS if present. Although TT frames have the highest priority, if an AVB or a BE frame is already in transmission at the beginning of time window for TT, TT traffic may be delayed depending on which *integration mode* is being used. (1) *Non-preemption* integration mode is defined in IEEE 802.1Qbv and uses a "guard band" to protect scheduled TT traffic, such that the gates associated with AVB and BE traffic are closed in advance during the "guard band" to make sure that the link is idle (no AVB or BE traffic is still transmitting) when a TT queue is open for transmission. (2) The *preemption* mode is defined by IEEE 802.1Qbu and will allow TT frames to interrupt AVB frames, which are resumed once the transmission of the TT frame completes.

The impact of integration modes on the AVB traffic has been investigated in [ZPZL18]. As the experimental results in that paper show, the non-preemption integration mode will lead to wasted bandwidth due to the guard band, but it ensures no delays for TT traffic. When preemption is used, the remaining AVB frame will include an overhead used to separate and reassemble at the destination ES. However, compared to the guard band, the overhead of 24 bytes is much smaller. Therefore, the use of preemption will decrease the latency of AVB traffic and improve bandwidth usage; this is at the expense of a slightly increased jitter for TT traffic. In this paper we consider the non-preemption integration mode (which favors TT traffic, see the discussion in section 3.7) and will not further investigate this aspect, which has been discussed in [ZPZL18].

3.4.1 TT Traffic

The Gates are opened and closed by a *Time-Aware Shaper* (TAS), according to a port-specific *Gate-Control List* (GCL) dictating the state of the gates at defined times relative to the start of the GCL. For example, in the GCL in Figure 3.3, $T001$:*cOcccccc* means that at time "$T001$" the TT queue 6, identified by using its

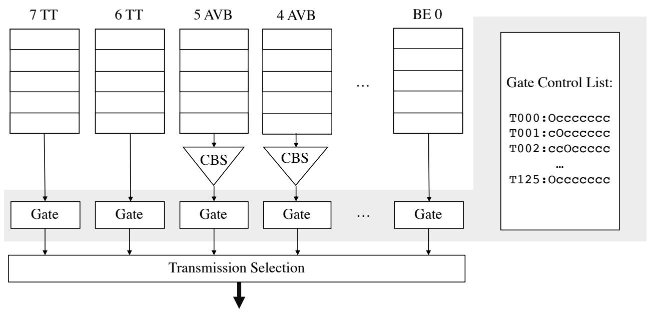


Figure 3.3: Example GCL configuration

queue priority as index, is open (O) and all the rest are closed (c). The start of these GCLs are synchronized across the bridges using the *time synchronization* defined in IEEE 802.1AS able to ensure a common view of time across the system. Each GCL is repeated with a period typically set to be a multiple of the *Least Common Multiple* of all the periods used in the system, denoted with T_{cycle}.

As suggested in IEEE 802.1Qbv, to completely avoid interference from other traffic-types, we assume the GCLs are constructed such that the TT queue is the sole available queue when open and all the remaining queues have been closed in advance. Thus, every time the TT transmission gate opens, the TT frames can be transmitted immediately and by the synchronization of the GCLs on the entire path from sender to receiver, a TT frame can be transmitted without having to be subjected to any queueing delays. This makes the TT traffic type suitable for jitter- and latency-sensitive applications.

Because we enforce determinism in the GCLs [CSCS16][SCS18], i.e., only frames of one of the messages are present in the queue at a time, we can model communication schedules at the frame-level. Thus, our TT model captures the scheduling of the frames in terms of an offset, period and duration; we use the notation $\mathcal{S}^{m_i}$ to capture the schedule table of the frames of a message m_i. For example, the scheduling for $m_1 \in \mathcal{M}^{\mathrm{TT}}$ with a route $[ES_1, BR_1], [BR_1, BR_2], [BR_2, ES_2]$ in 3.1a is modeled as $\langle$*offset, period, duration*$\rangle = \langle 0\mu s, 62.5\mu s, 10.4\mu s\rangle$, where the duration denotes the amount of time the TT queue has exclusive access to transmit the TT frame.

3.4.2 AVB Traffic

AVB has currently two traffic classes in TSN, AVB *Class A* and *Class B*, where Class A has higher priority and tighter timing requirements. However, the TSN task group is working towards supporting fully customizable AVB classes, allowing the definition of multiple AVB traffic classes. Hence, our model is general and considers an arbitrary number of AVB traffic classes, denoting an AVB class with M_i, and the set of available AVB classes with M^{AVB}. The AVB Class M_i has higher priority than the AVB Class M_{i+1}. The messages assigned the same priority belong to the same AVB traffic class M_i, and frames within each traffic class are forwarded in FIFO order. The number of AVB classes N_{AVB} (corresponding to the number of AVB queues) and the assignment of AVB messages to AVB queues $Q_{AVB}(m_i)$ (that decides their AVB class), is determined by our optimization approach.

An AVB frame is transmitted when (i) the gate of its queue is open, (ii) there is no other higher priority frame being transmitted and (iii) if its CBS allows it. The CBS standardized in IEEE 802.1Qav in conjunction with the amendments in IEEE 802.1Qbv makes the queue available for transmission whenever the amount of *credit* is positive or zero. The purpose of the CBS is to shape the transmission of AVB frames in order to prevent bursts and starvation of the lower priority queues. Credits are initially zero, they are decreased with a *send slope* while transmitting and frozen while the gate is closed. Transmission is only initiated when credit is non-negative. The credit is increased with an *idle slope* when frames are waiting, but they are not being transmitted. If the queue is emptied while the credit is positive, the credit is set to zero. The *idle slopes* and *send slopes* are configuration parameters for the AVB classes; there is one queue for each AVB class, and the slopes will affect all messages in the respective queue. The sub-standard IEEE 802.1Qbv constrains the send slope such that $send\ slope = idle\ slope - C$ where C is the physical link transmission rate.

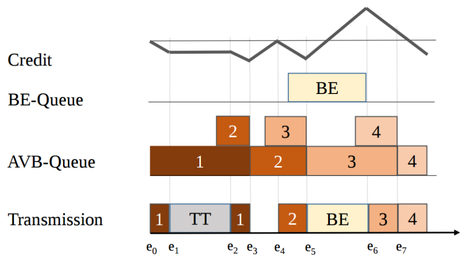


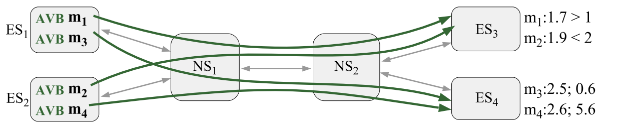
Figure 3.4: Example AVB transmission

An example of how the CBS works is illustrated in Figure 3.4 where we have a TT frame, one AVB queue that has to transmit frames, 1 to 4, as well as a BE queue. The figure shows a timeline for the transmission on the bus, where a rectangle is a part of a frame, with the width representing the transmission time. The AVB and BE queues show on the y-axis the number of queued frames, and on the x-axis the waiting time in the queue. The value of the credit over time is presented on the top of the figure. Let us explain the transmission of the the AVB frame in Figure 3.4 using the events (e_0) to (e_7) depicted on the bottom timeline:

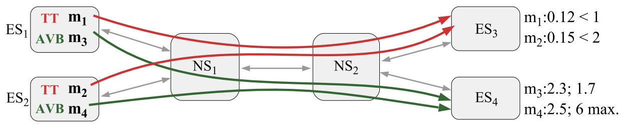
(e_0) AVB Frame 1 starts to transmit and the credits are decreased according to the send slope. (e_1) Let us assume that a TT frame is scheduled as depicted in the bottom timeline of Figure 3.4. The AVB queue is closed to make room for the TT transmission. AVB Frame 2 arrives and is enqueued while the credits are frozen. (e_2) The TT transmission finishes and the AVB-Queue opens and resumes the transmission of AVB Frame 1. During this transmission, the credits are decreased again. (e_3) Transmission of AVB Frame 1 finishes, but as the credit at this point is negative, AVB Frame 2 is not transmitted. Meanwhile AVB Frame 3 is enqueued and the credits are accumulating according to the idle slope. (e_4) Credits have increased to zero, hence AVB Frame 2 is transmitting. During this transmission, the credits are decreasing according to the send slope. During this time, a BE frame is enqueued. (e_5) The transmission of AVB Frame 2 finishes, and since the credit is negative, the lower prioritized BE frame is selected for transmission. AVB Frame 4 is enqueued and credits are accumulating. (e_6) The transmission of the BE Frame finishes and AVB Frame 3 is selected for transmission. (e_7) The transmission of AVB Frame 3 finishes and the excess credits accumulated transmitting the BE frame are used to immediately initiate transmission of AVB Frame 4.

3.5 Problem Formulation

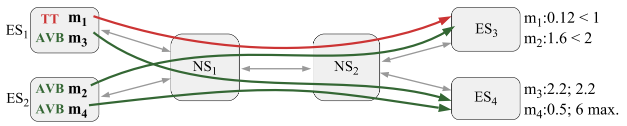
As an input to our problem we have the topology of the TSN-based network $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the set of messages $\mathcal{M} = \mathcal{M}^{HRT} \cup \mathcal{M}^{SRT} \cup \mathcal{M}^{NC}$; for each message we know its parameters, as described in section 3.3, including the routing, and the set of available AVB classes M^{AVB}. We are interested to determine a network configuration Ψ such that all HRT messages are schedulable and the total utility for SRT messages is maximized. Deciding on a network configuration Ψ means determining, for each message $m_i \in \mathcal{M}$ the traffic type $\mathcal{TC}(m_i)$. For each TT message m_i, we also decide the schedule tables $\mathcal{S}^{m_i}$. We also decide the number of TT queues N_{TT} and the assignment Q_{TT} of TT messages to TT queues. For each AVB message m_j we decide (3) their AVB Class $M^k, M_k \in M^{AVB}$. There



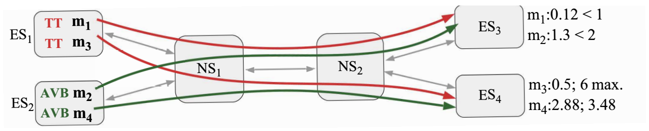
(a) All messages are AVB; m_1 is **not** schedulable; total utility is only 6.2 out of 12.



(b) HRT messages are TT and SRT are AVB. m_1 and m_2 are schedulable, but the total utility is 7.7 out of 12.



(c) HRT m_2 is AVB, but still schedulable; the total utility is 8.2, and m_3's utility is increased to 2.2



(d) HRT m_2 is AVB, SRT m_3 is TT. HRT are schedulable, and the total utility is increased to maximum of 12.

Figure 3.5: Motivational example

is one AVB queue for each AVB Class and thus the assignment $Q_{AVB}(m_j)$ of an AVB message m_j to its queue is decided based on its AVB Class. For each AVB Class M^k we also (4) decide the M_k.*idle slope* and implicitly its *send slope*.

Let us consider the architecture and application from Figure 3.1, where TSN is used as communication protocol. As mentioned, the maximum utility of both SRT m_3 and m_4 is 6, so the total maximum utility achievable is 12. We are interested to determine the traffic type for each message; in this example, we consider for AVB a single AVB Class M_1, having the idle slope of 75% out of the link speed C (this is the default maximum slope value of AVB Class A in the current standard). In the following examples for different traffic type assignment $\mathcal{TC}$, we determine for this small example, the optimal schedule table $\mathcal{S}$, i.e., such that the utility of SRT messages is maximized and the HRT messages are schedulable. For the AVB frames, we use the Worst-Case end-to-end Delay

(WCD) analysis from subsection 3.6.3 to determine their WCD R_i. For the TT frames, the WCD is derived directly from the schedule tables, as the time when the frame is received at its destination, relative to its sending time. A HRT frame is schedulable if its WCD is lower or equal to the deadline, and the utility of a SRT frame m_i is given by $m_i.utility(WCD)$, as specified in 3.1b.

Multiple traffic types are necessary to support mixed-criticality applications. For example, if we do not have the TT traffic type, and make all messages AVB, we obtain the solution depicted in 3.5a. In all the examples in Figure 3.5 we indicate next to the message source the traffic type used; we depict the respective route with green for AVB and red for TT. We write next to the destination of each HRT message its WCD and compare it to its deadline; next to each SRT message destination we have its WCD, followed by its utility. As we can see from 3.5a, if all frames are AVB, then m_1 misses its deadline, i.e., $1.7 > 1$, and the total utility for SRT messages is only 6.2 out of the maximum of 12.

A possible solution would be to use TT for HRT messages and AVB for SRT, as depicted in 3.5b. Such an approach is used implicitly, for example, in the context of TTEthernet in [TSPS15], which does not attempt to optimize the traffic types. As expected, by using the TT traffic type for HRT messages m_1 and m_2, we can make them schedulable, since we can synthesize the TT schedules such that the HRT messages have a very low latency. However, as discussed in section 3.4, TT frames have the highest priority, and when doing the traffic integration (non-preemption mode is considered in this example), the AVB frames may be delayed by the TT frames. Due to these delays, the utility of SRT message m_3 is only 1.7 even if m_4 has a maximum utility. Recall that schedules are optimal with respect to HRT schedulability and SRT utility; in this case, delaying the TT frames will not help the AVB frames because of the *non-preemption* integration policy, which does not allow any AVB frame to start its transmission if it may delay a TT frame.

By using the AVB traffic type for the HRT message m_2 instead of TT, we will get in 3.5c a larger WCD for m_2, of 1.6 ms instead of 0.15 ms in 3.5b. However, since $m_2.deadline = 2$, m_2 is still schedulable. The utility of m_3 becomes a bit higher, namely 2.2. If we further optimize the traffic type assignment, and we modify the solution in 3.5c to change the traffic type of SRT m_3 from AVB to TT (as depicted in 3.5d), we are able, by carefully deciding on the schedule table for the TT frames, to reduce its WCD and thus increase m_3's utility also to the maximum of 6.

As this motivational example shows, only by optimizing the assignment of traffic types for the mixed-criticality frames, we are able to obtain good quality solutions, which guarantee the schedulability of HRT messages while maximizing the utility for the SRT messages. Note that making all the frames TT regardless of their timing criticality could also be a solution. However, in practice, legacy non-scheduled traffic has to be integrated in the system, and the system engineer may prefer that some frames are AVB for flexibility. Updating schedules to accommodate new messages may trigger re-validation activities, which are costly. In addition, as the number of frames increases (systems may have tens of thousands of frames, even millions of frames [PSRH15]), the ESes and NSes will run out of memory for the required schedule tables. Although methods such as [PSRH15] can handle a large number of TT frames, they are not able to integrate the schedulability analysis of non-scheduled frames, see section 3.8 for a discussion.

3.6 Optimization Strategy

The problem presented in the previous section is NP-complete [Ull75]. Our problem includes also the scheduling problem (for TT messages), the later one being equivalent to the NP-hard flowshop scheduling problem [GJS76]. We propose a Tabu Search (TS)-based metaheuristic solution, called Traffic Type Assignment (TTA) to solve this optimization problem. TS-based approaches have been shown to produce good quality results for the problem of optimizing the configuration of TTEthernet systems [TSPS15]. Our TTA, presented in Algorithm 3.1, takes as input the network topology $\mathcal{G}(\mathcal{V}, \mathcal{E})$, the set of messages in the application $\mathcal{M} = \mathcal{M}^{HRT} \cup \mathcal{M}^{SRT}$, including the properties of each message, the legacy traffic type assignment $\mathcal{TC}^0$, i.e., some messages may already be assigned a fixed traffic type, and the set of available AVB classes M^{AVB}. TTA produces as output an implementation Ψ, which contains the traffic type assignment $\mathcal{TC}$, the schedule tables $\mathcal{S}$ for TT messages, the AVB Class M_i for the AVB messages and the corresponding $M_i.idle\ slope$ for each AVB Class.

TS metaheuristics [BK14] search for that solution which maximizes a *quality function*. The quality function used to evaluate a generated solution is presented in subsection 3.6.2. TS is based on a neighborhood search technique, where the current solution is modified using design transformations (also called *moves*) to generate neighboring solutions. The moves we propose, including an example of how TS works for our problem, are presented in subsection 3.6.4.

In each iteration (lines 3—18) TS generates and evaluates multiple solutions. At the beginning of each iteration TTA creates a neighborhood of the current solution containing maximum N neighboring solutions (line 4). The function *SelectCurrent* (in line 5) returns either the neighboring solution which is not "tabu" and which maximizes the quality function or the empty solution $\emptyset$. To avoid revisiting recently explored solutions, TS keeps a *tabu list* T, which is

Algorithm 3.1 $TTA(\mathcal{G}(\mathcal{V},\mathcal{E}),\mathcal{M},\mathcal{TC}^0,\mathcal{S}^0,M^{AVB})$

```
1: $current \leftarrow best \leftarrow GenerateInitial(\mathcal{G}(\mathcal{V},\mathcal{E}),\mathcal{M},\mathcal{TC}^0,\mathcal{S}^0,M^{AVB})$
2: $T \leftarrow \emptyset$
3: repeat
4:    $neighbors \leftarrow GenerateLimitedNeighborhood(current, N)$
5:    if $\emptyset \neq current \leftarrow SelectNeighbor(neighbors, T)$ then
6:       $UpdateTabu(T, current, K)$
7:    end if
8:    if $Cost(current) > Cost(best)$ then
9:       $best \leftarrow current$; $stagnation \leftarrow 0$
10:   else
11:      $stagnation{+}{+}$
12:   end if
13:   if $stagnation = \Omega$ then
14:      $current \leftarrow DiversifyCurrent(current, T)$
15:   else
16:      $MarkIteration(T)$
17:   end if
18: until time limit reached
19: return best
```

a selective history of solutions that have already been visited (line 6). Each solution is considered "tabu" at most K iterations.

If the currently explored solution is better than the best known solution (line 8), it is saved as the "best-so-far" solution (see line 9). If the current solution does not represent an improvement (line 10), the *stagnation* counter is increased (line 11).

To avoid getting trapped in a local optima, TS uses "diversification" (line 14), i.e., forcing the algorithm to look in unexplored areas. The diversification method we use is applied when no improvements are observed after a given number of Ω iterations (line 13) and consists in switching the traffic type to a randomly selected set of non-legacy messages. If no diversification is required the current iteration is marked in the tabu list T (line 16). The *MarkIteration* modifies T such that: (1) the number of iterations that has to remain "tabu" is decreased for each solution and (2) those solutions for which the number of iterations becomes 0 are removed from T. Our TTA stops when a given time-limit has been reached.

3.6.1 Initial Solution

TS can start from any initial solution, including a random solution. Our initial solution was constructed such that it helps TS to converge faster to a good quality solution. The initial solution of our TTA implementation is listed in Algorithm 3.2. We consider shortest path routing for all routes. For the traffic type assignment $\mathcal{TC}$ we consider that HRT messages are TT and SRT messages are AVB, under the constraints imposed by the given $\mathcal{TC}^0$. We use the method in [RP17] to generate the schedule tables $\mathcal{S}$ for the TT messages, considering the schedules $\mathcal{S}^0$ for the legacy TT messages. All AVB messages are assigned to the "nearest" AVB class. The nearest AVB class for a message is the class that minimizes the absolute difference between message maximum bandwidth and the AVB class idle slope. We would like to recall that for any message $m_i \in \mathcal{M}$ the maximum bandwidth is given by the ratio $m_i.size/m_i.period$. The initial $M_i.idle\ slope$ for each AVB class M_i is assigned such that it avoids creating an infinite backlog of AVB frames, i.e., the idle slope should be at least equal to the maximum accumulated rate of AVB class M_i flows, see [ZPZL18] for a discussion. The $M_i.send\ slope$ is $M_i.idle\ slope - C$.

Algorithm 3.2 $GenerateInitial(\mathcal{G}(\mathcal{V},\mathcal{E}), \mathcal{M}, \mathcal{TC}^0, \mathcal{S}^0, M^{AVB})$

```
1: for all $m_i \in \mathcal{M} \backslash \mathcal{TC}^0$ do
2:   if $m_i \in \mathcal{M}^{SRT}$ then
3:     $\mathcal{TC}(m_i) \leftarrow AVB$
4:     $m_i.AVB \leftarrow NearestClass(M^{AVB}, m_i)$
5:   end if
6:   if $m_i \in \mathcal{M}^{HRT}$ then
7:     $\mathcal{TC}(m_i) \leftarrow TT$
8:   end if
9: end for
10: $\mathcal{S} \leftarrow ScheduleAndPostprocess(\mathcal{G}(\mathcal{V},\mathcal{E}), \mathcal{M}{\cdot}\mathcal{S}^0, \mathcal{TC}^0)$
11: return $\mathcal{TC}(\mathcal{M}), \mathcal{S}, M^{AVB}\mathcal{M}^{AVB}$
```

The function *ScheduleAndPostprocess* creates the schedule tables for TT messages using the approach from [RP17], which employs a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic. GRASP-based scheduling method is able to handle large problem sizes and produce good quality solutions within a short runtime. However, GRASP attempts to optimize the TT schedules by packing them as much as possible. This will lead to scheduling TT frames back-to-back in large blocks, which would introduce very large delays for AVB messages.

Hence, after synthesizing the schedule tables we are interested to process these initial schedules (line 10 in Algorithm 3.2) such that we introduce space in the schedules for AVB messages. Note that during the search procedure of Tabu Search, this space will be automatically introduced by the "Modify GCL" move, see subsection 3.6.4 for details.

Thus, for each link, (1) a TT blocks structure (*tbs*) is determined and (2) the messages in the created schedules are redistributed based on *tbs*, i.e., we schedule earlier or later time windows of the schedule tables $\mathcal{S}$ without affecting the legacy schedules $\mathcal{S}^0$, conserving the order of the messages sent on a link. The *tbs* structure is built for each link based on the maximum transmission times of AVB messages, considering their size, on that link.

3.6.2 Quality Function

We evaluate each solution Ψ visited by the Tabu Search using the following quality function:

$$Cost(\Psi) = wp_{HRT} \cdot \delta_{HRT} + \sum_{m_i \in \mathcal{M}^{SRT}} m_i.utility(WCD(m_i)) \tag{3.1}$$

where the first term represents a constraint which checks for the schedulability of HRT messages, and the second term is the total utility of SRT messages. δ_{HRT} captures the "degree of schedulability" of a solution and is defined as

$$\delta_{HRT} = \sum_{m_i \in \mathcal{M}^{HRT}} \min(0, m_i.deadline - WCD(m_i)) \tag{3.2}$$

where $WCD(m_i)$ is the worst-case end-to-end delay of the HRT message, calculated as presented in subsection 3.6.3. Note that δ_{HRT} will be zero in case all HRT messages are schedulable, i.e., WCD is smaller than the deadline, otherwise it is a negative value. We multiply δ_{HRT} with a penalty value wp_{HRT}, which is two times greater than the value of the maximum total utility. If HRT messages are schedulable and thus δ_{HRT} is zero, the first term in Equation 3.1 will not contribute to the quality function, and the search will attempt to maximize the total utility (the second term). However, if HRT messages are not schedulable, the penalty value will push TTA to search for schedulable HRT solutions.

3.6.3 WCD Analysis

The Worst-Case end-to-end Delay $WCD(m_i)$ of a message m_i is calculated differently depending on its traffic type, as we discuss next. For all messages, we take into account for the possible fragmenting of an AVB message into several frames. Let R_{m_i} be the WCD determined for the last frame $f_{i,k}$ of m_i. Then, the WCD of m_i is $WCD(m_i) = m_i.period \cdot (k-1) + R_{m_i}$, where $m_i.period$ is the period of the frames of message m_i. Note that the analysis of R_{m_i} accounts for the multiple destinations of m_i, taking the largest WCD over the destinations.

TT: Recall that TT frames are sent based on schedule tables. For a TT message m_i packed into a set of frames $\mathcal{F}_i$(with $f_{i,x}$ denoting the x^{th} frame instance) that are sent from a source ES_i^{src} to multiple destinations $\mathcal{ES}_i^{dest}$, the WCD being the maximum time in the receiving schedules of the destination ESes. The WCD is described in Equation 3.3 and captures the time the last frame is received at its destination, relative to the sending of the first frame at the source ES.

$$WCD(m_i) = \max_{f_{i,n}, ES_l \in \mathcal{ES}_i^{dest}} S_{R,ES_l,n} - S_{S,ES^{src},1} \tag{3.3}$$

AVB: There have been several WCD analysis methods proposed for AVB frames, see [ZPZL18] for the related work in this area. Although latency analysis methods have been successfully applied to AVB traffic in AVB networks, e.g., [DAB14], they do not consider the effect that TT traffic has on the AVB traffic in TSN. A Network Calculus-based analysis to compute the WCDs of Rate-Constrained (RC) traffic with the consideration of the scheduled TT frames in TTEthernet has been proposed [ZPL+17], but the technique is not applicable to TSN, see subsection 3.1.1 for a discussion.

Recently, the AVB Latency Math equation from the sub-standard 802.1BA has been extended to consider the TT traffic in TSN [Lau16]. However, it can only be used for AVB Class A traffic and, as demonstrated in [ZPZL18], the analysis is both unsafe and overly pessimistic. An interval-based analysis is proposed in [MS17], where the end-to-end delay is computed by summing up the independently obtained worst-case per-hop delay. However, the analysis in [MS17] is pessimistic due to the fact that the per-hop delays are computed neglecting the whole path.

Hence, in this paper, we have integrated the timing analysis for AVB traffic in a TSN network from [ZPZL18], which is currently the only approach for TSN that considers TT traffic. The authors use a Network Calculus-based method to determine the WCDs of AVB flows in a TSN network, considering the effects of TT traffic controlled by GCLs, guard bands for the non-preemption mode and preemption overheads for the preemption mode.

3.6.4 Tabu Search Moves and Example

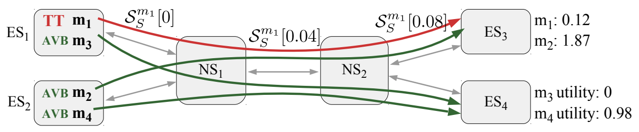
Next we are going to present the moves used to generate the neighborhood of a solution. A neighboring solution is obtained by applying randomly one of the following three moves: (1) *Switch Traffic Type* (STT), (2) *Modify GCL* (MGCL) or (3) *Modify Class* (MC).

When switching the traffic types of TSN, namely TT and AVB, the following rules are considered: (i) the STT move can be applied to all messages, except the legacy messages from $\mathcal{TC}^0$, (ii) when switching an AVB message to TT, we compute the schedule table using the same method as mentioned for the initial solution and (iii) when switching from TT to AVB the message assignment to AVB classes is similarly as in the initial solution.

The MGCL move will postpone or advance the offsets of the TT frames, see subsection 3.4.1. First, we randomly select a frame of a TT message and identify its route r over which the message is routed. Next, the offset of the selected frame is modified on each of the dataflow links of its route r. Modifying an offset of a frame means: postponing it, or scheduling it later, or advancing it, or scheduling it earlier. Since modifying the offset of a TT frame may impact the schedule of other TT frames, after applying the MGCL move we should recheck the validity of schedule tables for all frames affected by this move.

In TSN for AVB traffic the parameters that affect indirectly the messages waiting time in the queue are the idle and send slope. Therefore, although in TSN we cannot modify the shaping parameters for individual AVB messages, we can control indirectly the latency of AVB messages by assigning them to different AVB classes. As we saw previously, an intuitive assignment is to assign a message to the nearest AVB class, but still, if too many messages that are sharing a link are assigned to the same AVB class then the waiting time per message is increased. Therefore, by using the MC move we allow an AVB message with class M_i to be assigned to another AVB class (randomly choosing between M_{i-1} and M_{i+1}). With a given probability (determined experimentally), the MC move also increases or decreases the $M_i.idleslope$ of an AVB class M_i under the constraints defined in [ZPZL18] such that it avoids an infinite backlog of AVB frames. Note that a higher value for the idle slopes would allow the AVB flows to pass faster through the switches, reducing their WCDs.

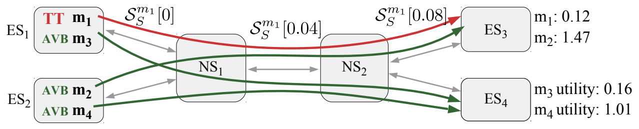
Let us illustrate how TTA works. Let us consider the example from Figure 3.1, and let us assume that the *current* solution is the solution depicted in 3.6a,



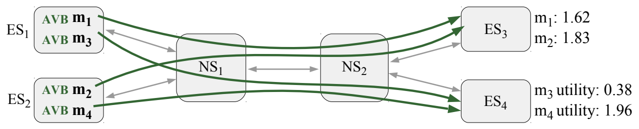
(a) The *current* solution is the example from 3.5c; Quality=0.98

Message	$\mathcal{TC}$	link	$\mathcal{S}$/*idle slope*	iterations
m_1	TT	$NS_1 - NS_2$	[0.09]	14
m_2	AVB	—	M_2	5
m_3	TT	$ES_1 - NS_1$	[1]	0
m_3	TT	$NS_1 - NS_2$	[1.3]	7

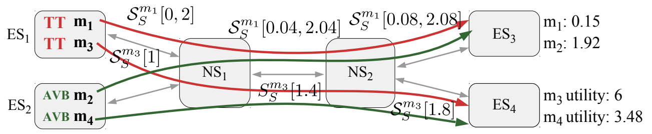
(b) Tabu list



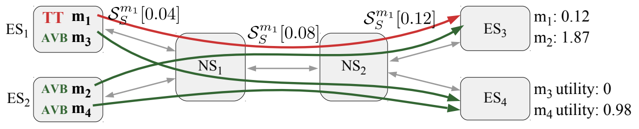
(c) Modify Class: M_2.*idle slope* = 50% assigning m_2 to an AVB class which halves *idle slope*; *Cost* = 1.17; tabu



(d) Switch Traffic Type of m_1 from TT to AVB; *Cost* = −2.62; non-tabu



(e) Switch Traffic Type of m_3 from AVB to TT; *Cost* = 9.48; non-tabu



(f) Modify GCL of m_1 on $ES_1 - NS_1$ by postponing it with 0.04 ms; *Cost* = 0.98; non-tabu

Figure 3.6: Example TS neighborhood search

which is also the *best-so-far* solution. Similar to the examples in Figure 3.5, we denote the traffic type next to the message source, and the WCD and utility values next to the destination of the message. In addition, we also show, for the TT messages, the schedules $\mathcal{S}$ for each of the dataflow links where these are transmitted. For example, in 3.6a m_1 is sent on the last link, from NS_2 to ES_3 at time 0.08 ms. For simplicity, in this example, we use a single AVB class M_1 and its *idle slope* is set to 100%.

Recall that TTA uses a *tabu list* to avoid revisiting recently visited solutions. The tabu list for 3.6a is presented in 3.6b, and stores on each row information about a particular solution visited in the past. Instead of storing the complete solution, we only store information related to the move that has generated the solution, i.e., the transformations performed. Thus, we store the message involved, the dataflow link (for TT frames) and the schedules $\mathcal{S}$ for the TT frames and *idle slope* for the AVB frames. In the last column, we store the number of iterations this solution has been considered tabu. This value starts at the "tabu tenure" K, which we set to 25 (line 6 in Algorithm 3.1), and is decremented every iteration if there is no diversification required (line 16 in Algorithm 3.1). Let us remind that *MarkIteration* also removes from the tabu list the entries whose "iterations" became 0, see the corresponding row of m_3 in the table in 3.6b.

As a first step, we generate from the *current* solution the neighborhood solutions using the moves presented earlier. Since the neighborhood can be quite large, we restrict the neighborhood *neighbors* to a maximum number of N solutions. We use $N = 7$ in our experiments, but for the sake of simplicity in this example let us assume that $N = 4$. Thus, the *neighbors* are obtained by randomly applying the moves on the *current* solution, obtaining the neighboring solutions in Figure 3.6c–f. For each candidate solution we write the following in its caption: the move that has generated it, the value of the *Quality* function, for which we considered in this example a penalty $wp_{HRT} = 8$, and if the move is tabu or not.

TTA will select that neighbor which improves the quality function and it is not tabu. For the neighbors in Figure 3.6c–f, the neighbor in Figure 3.6c is not replacing *current* because it is contained in the tabu list. The other candidates Figure 3.6d–f are not in the tabu list (note that we have removed the third row in the tabu list in 3.6b—see line 16 in Algorithm 3.1). TTA will select that solution, which maximizes the quality (in our case, Figure 3.6e), to replace *current* and the *best-so-far* (since that one is improved as well). The search is continued from the new *current* solution in a similar way.

3.7 Experimental Evaluation

For the evaluation of our *Traffic Type Assignment* (TTA) optimization strategy we used five synthetic test cases, *TC1* to *TC5* and two real-life case studies, *SAE* and *CEV*. CEV is the “Orion Crew Exploration Vehicle (CEV)” case study from [TSPS15], and SAE is the “SAE automotive communication benchmark” [Veh93], both adapted to use TSN. TTA was implemented in Java (JDK1.8) and all experiments were run on Intel Xeon E5-2665 machines at 2.4 GHz. We consider the non-preemption integration mode (see section 3.4) and we have ignored the BE traffic, hence we have set the idle slopes for AVB to 100%.

In the first set of experiments we were interested to determine the quality of our TTA algorithm compared to the setup when all messages are AVB. We have not considered legacy messages in this first set of experiments. Thus, we have compared the results obtained with TTA to *AVB*, which simply uses AVB for all messages. The algorithm “*AVB*” is also implemented as a Tabu Search, where we have removed all TT-related moves, and instead only use moves that change the AVB parameters.

The results are presented in Table 3.1. The number of ESes and NSes in the architecture, as well as number of HRT and SRT messages in the applications, are presented in columns 2–5, respectively, in Table 3.1. We have run both solutions, TTA and *AVB* on the test cases in Table 3.1, and we show for both two values, for each test case: the percentage of HRT messages found schedulable (HRT sched.), and the percentage of total utility of SRT messages, compared to the maximum utility achievable (SRT util.). The time limit used for TTA and *AVB* for each test case is given in column 8 in hours and minutes.

As we can see from the table, *AVB* is unable to obtain schedulable solutions (for 4 out of 5 test cases only about half of the HRT messages are schedulable), and the utility of the SRT messages is lower compared to TTA. By optimizing the assignment of traffic classes to mixed-criticality messages, we were able to obtain with our TTA schedulable solutions in most cases (100% schedulable

ID	Arch.		Applications		Runtime	AVB		TTA	
	ES	NS	HRT	SRT	h:min	HRT sched.	SRT util.	HRT sched.	SRT util.
TC1	8	3	9	11	0:50	44.44%	90.27%	100%	100%
TC2	8	3	11	23	2:30	54.54%	85.07%	100%	99.63%
TC3	8	3	17	28	3:45	47.06%	64.10%	100%	95.77%
SAE	15	7	40	39	5:00	70.00%	81.72%	100%	94.61%
Orion	31	15	99	87	12:30	45.45%	78.80%	94.94%	98.68%

Table 3.1: Comparison of “AVB only” with an optimized assignment of TT and AVB using TTA

HRT messages), or very close to full 100% schedulability. TTA is also able to significantly improve the utility compared to *AVB*, from 64.10% to 95.77% utility in the case of *tc3*. As it can be observed TTA scales well with the size of the system (network and applications), being able to obtain good quality results also for the larger case studies.

We were also interested to compare TTA with the optimal solution. Due to the complexity of the problem, we were able to run an exhaustive search to get the optimal solution only for the smaller test case *tc1*. TTA has also been able to find the optimal result for this case, after a runtime of 50 minutes.

In the second set of experiments, we were interested to compare our proposed TTA solution to the situation when we use only the TT traffic class for messages. We assume that the GCLs are synthesized using the approach from [PSRH15] [CS16] that uses Satisfiability Modulo Theories (SMT), which we have modified to consider the TSN determinism constraints, see subsection 3.4.1. The proposed SMT approaches use a decomposition strategy, where the problem is decomposed into multiple sub-problems allowing the scheduling of tens of thousands of TT frames (with the caveat that the systems should not have a high utilization). The SMT synthesis approach for TT was implemented in C (GCC5.4) and was run on an Intel Core i7-5600U machine at 2.6 GHz.

The results are presented in Table 3.2. For this comparison we used two synthetic test cases, *TC4* and *TC5* and one realistic case study. Since the existing SMT formulations can only handle star topologies, we have modified the topologies of these test cases to use a star topology, and the number of ESes and NSes in the architecture are presented in columns 2–3, respectively. Thus, the SAE test case mentioned in the first set of experiments becomes here *SAE2*. In this set of experiments, we do not ignore the legacy messages. Thus, the "Applications" columns show both "Non-legacy" messages for which we need to determine the traffic class, and "Legacy" messages for which the traffic class is already decided and cannot be changed. We can see for each category the Total (tot.) number of messages, broken down into Hard Real-Time (HRT) and Soft Real-Time (SRT) messages. We have used the "Initial solution" from Algorithm 3.2 to configure the legacy TT and AVB messages.

Our TTA approach optimizes the traffic class for all non-legacy messages, deciding between TT and AVB. However, the SMT approach uses the TT traffic class for all non-legacy messages. The SMT cannot integrate an impact analysis of TT schedules on AVB (it is no longer scalable), hence in the SMT solution, the legacy AVB messages are added afterwards on top of the created schedules.

Table 3.2 shows the obtained results under the "SMT" and "TTA" headings. For both legacy and non-legacy messages we show, as in Table 3.1 the percentage

ID	Arch.		Applications						SMT				TTA			
			Non-legacy			Legacy			Non-legacy		Legacy		Non-legacy		Legacy	
	ES	NS	tot.	HRT	SRT	tot.	HRT	SRT	HRT sched.	SRT util.	HRT sched.	SRT util.	HRT sched.	SRT util.	HRT sched.	SRT util.
TC4	7	1	21	7	14	9	4	5	100%	100%	0%	60%	100%	100%	100%	100%
TC5	15	1	50	19	31	16	5	11	100%	100%	10%	0%	100%	100%	100%	82%
SAE2	15	5	60	21	39	20	5	15	100%	100%	40%	79%	100%	95%	100%	93%

Table 3.2: Comparison of our TTA approach with SMT-based solvers in the presence of legacy messages

of HRT messages found schedulable (HRT sched.), and the percentage of total utility of SRT messages, compared to the maximum utility achievable (SRT util.). Having all messages (both HRT and SRT) as TT allows the designer greater control over their WCDs and jitters by carefully creating the schedules. Hence, as expected, the SMT approach can obtain a 100% schedulability for HRT and 100% utility for SRT. However, when making all messages TT, the challenge is to integrate the legacy messages. As we can see in Table 3.2 under the "Legacy" heading for the SMT approach, the HRT schedulability and SRT utility of legacy messages is very low. However, by using our TTA approach, we are able to reassign some of the messages from the TT traffic class to AVB in order to accommodate the requirements of the legacy messages. As we can see from the columns labeled "Legacy" for TTA, we are able to dramatically increase the HRT schedulability and SRT utility without reducing the performance for the non-legacy messages: only the utility of non-legacy SRT messages for the SAE test case is slightly reduced from 100% to 95%, an acceptable penalty to pay in order to find a good solution that does not impair the legacy messages.

3.8 Discussion and Conclusions

In this paper we have considered mixed-criticality applications, using hard real-time and soft real-time messages, implemented on TSN-based distributed cyber-physical systems. We have used a hard deadline for the HRT messages and a utility function for SRT messages. We have proposed a Tabu Search-based metaheuristic, which we called Traffic Class Assignment (TTA), to determine the assignment of traffic classes (Audio Video Bridging, AVB, and Time-Triggered, TT) to the mixed-criticality HRT and SRT messages. TTA also optimizes the schedules for the TT frames and the AVB parameters (AVB Classes and slopes).

Researchers have debated, in the cyber-physical real-time systems area, the time-triggered (TT) and event-triggered (ET) approaches. The advantage of a TT approach is that, by determining the schedule tables at design time, we have a fine-grained control of timing properties, e.g., we can reduce WCD and jitter.

Since the schedule synthesis is a non-polynomial hard problem, the challenge is the availability of a scalable scheduling algorithm. Realistic applications can have tens of thousands of messages. In TSN, the size of the GCLs is also limited by the switch implementation, which means that it is unrealistic to assume that all TT messages can be accommodated via GCLs.

ET is known to be more flexible and scalable, and TT may require a full redesign even for small changes or upgrades. In addition, as our experiments show, even the introduction of a small fraction of AVB legacy messages can significantly reduce the quality of TT-only solutions. Another drawback of TT is that it cannot easily accommodate aperiodic traffic or traffic that has varying payloads. In these cases, the option is to use a minimum inter-arrival time and consider the largest possible payload, which results in an overdesign that wastes the available bandwidth.

As mentioned, we have considered the non-preemptive integration policy, which favors TT messages by preventing the transmission of AVB messages that could delay TT transmission. As reported in [ZPZL18], using a preemptive integration policy, which allows AVB messages to be retransmitted once interrupted by TT messages, improves the WCDs of AVB messages and the overall bandwidth utilization at the expense of TT jitter. In such a context, we expect that using the AVB traffic class for some messages would bring additional benefits, since we could further trade-off the WCDs and jitter of TT to improve the timeliness of AVB.

Therefore, for real-time applications that do not require very tight WCDs and can tolerate non-zero jitter, it is worthwhile considering using the AVB traffic class. In this case, the challenge, as we have discussed in this paper, is to decide the appropriate traffic class. As the experimental results show, our proposed TTA approach is able to determine, in a reasonable time, schedulable solutions (HRT messages meet their deadlines) which also improve the overall utility of the SRT messages.

Chapter 4

Paper C: AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN

IEEE 802.1 Time-Sensitive Networking (TSN) is a set of amendments to the IEEE 802.1 standard that enable safety-critical and real-time behaviour over Ethernet for the industrial automation and automotive domains. Selected TSN mechanisms offer the possibility to emulate the well-known traffic classes found in mixed-criticality distributed systems: Time-Triggered (TT) communication with low jitter and bounded end-to-end latency, Audio-Video-Bridging (AVB) streams with bounded end-to-end latency, as well as general Best-Effort messages, which have no timing guarantees. Critical traffic is guaranteed via the global network schedule which is stored in so-called Gate Control Lists (GCLs) and controls the timely behavior of frames for each queue of an egress port. Although researchers have started to propose approaches for the routing and scheduling (i.e., GCL synthesis) of TT traffic, all previous research has ignored lower priority real-time traffic such as AVB, resulting in TT configurations that may increase the worst-case delays of AVB traffic, rendering it unschedulable. In this paper, we propose a joint routing and scheduling approach for TT traffic, which takes into account the AVB traffic, such that both TT and the AVB

traffic are schedulable. We extensively evaluate our approach on a number of synthetic as well as realistic test cases.

4.1 Introduction

In this paper, we are interested in safety-critical and real-time applications implemented using distributed cyber-physical systems. Several real-time capable communication protocols have been proposed and are in use in different application areas, e.g., FlexRay for automotive [Con06], ARINC 664 p7 for avionics [Aer09], and EtherCAT for industrial automation [JB04]. However, emerging applications, e.g., Advanced Driver Assistance Systems (ADASes), autonomous driving, or industrial automation, have increasing bandwidth and real-time demands. For instance, autonomous driving requires data rates of at least 100 Mbps for graphical computing based on camera, radar, and Light Detection And Ranging (LIDAR) data, whereas CAN and FlexRay only provide data rates of up to 1 Mbps and 10 Mbps, respectively.

The well-known networking standard IEEE 802.3 Ethernet [IEE12] meets the emerging bandwidth requirements for a wide range of application areas, while remaining scalable and cost-effective. It does, however, lack real-time and dependability capabilities [Dec05]. Many extensions, such as EtherCAT, PROFINET, ARINC 664p7 [Aer09], and TTEthernet [SAE11], have been suggested and are used in the industry. Although they satisfy the timing requirements, they are incompatible, hence, the interoperability within the same network is not possible without losing real-time guarantees [DN16]. To mitigate this drawback, the IEEE 802.1 Time-Sensitive Networking (TSN) Task Group [TSN12] has been working on defining standard amendments for real-time and safety-critical enhancements over Ethernet.

In standard switched Ethernet networks, in which end-systems are interconnected through a series of physical links and bridges (switches), communication from one sender to one or multiple receivers (flows/streams) is done via frames that are forwarded via a route through the network. Standard switching fabric contains queues on the egress ports of the switches (and end-systems) which implement a standard priority scheme and which store frames until the port is free for transmission. Hence, a frame might experience queueing delay while waiting for the transmission of higher priority frames and earlier arriving frames with same priority. This leads to network congestion causing nondeterministic behavior and variance in frame arrival times.

TSN: First, IEEE 802.1Q-2005[1] introduced support for prioritizing the Best-Effort (BE) traffic providing some higher Quality-of-Service (QoS) properties. The IEEE Audio-Video Bridging (AVB) Task Group [AVB11] has developed another enhancement generally known as AVB which introduce two new shaped AVB traffic classes enabling bounded latency. In 2012, *TSN Task Group* has started extending the protocol towards safety-critical and time-sensitive applications which require even more stringent real-time guarantees. Via selected amendments defined in TSN, i.e., IEEE 802.1Qbv *Enhancements for Scheduled Traffic* and IEEE 802.11ASrev *Clock synchronization*, the well-known Time-Triggered (TT) traffic class, which has strict jitter and end-to-end latency guarantees, can be emulated in standardized TSN networks.

TT traffic requires schedule tables, called *Gate Control Lists* (GCLs), that defines the exact queue transmission times of frames on every egress port along the route of the respective flows. The schedules define at which points in time a so-called timed-gate that is associated with every queue is opened and when it is closed, enabling and disabling frame transmission, respectively. Since the schedules in different devices need to be aligned, a clock synchronisation mechanism is required in order to provide a global time reference. This synchronization protocol is defined in the IEEE 802.1ASrev standard and, together with IEEE 802.1Qbv, provide the basic building blocks for achieving determinism and bounded end-to-end latency for critical traffic. **AVB traffic** is intended for applications that require bounded end-to-end latencies, but that do not have the same stringent real-time requirements as TT traffic. In order to prevent the starvation of lower priority messages AVB introduces two new shaped traffic classes (AVB Class A and B) and uses the Credit-Based Shaper (CBS) defined in IEEE 802.1BA. The worst-case end-to-end delays (WCDs) of AVB flows can be analyzed via timing analysis methods, e.g., based on Network Calculus. Finally, **BE traffic** has the lowest priority and does not provide any timing guarantees.

Note that for implementing real-time applications, both TT and AVB traffic types provide bounded latencies, and the choice of traffic type for a message depends on the particularities of the application. The problem of determining the appropriate traffic type per message has been addressed in [GP16] for mixed-criticality traffic in TTEthernet. The WCDs of TT flows are determined by the GCLs. However, the WCD of an AVB message depends on the GCLs of TT traffic and the other AVB messages that share the same queue or have a higher priority. Recent timing analysis work for AVB [ZPZL18] has shown how to determine the WCDs of AVB messages taking into account the impact of TT traffic via the GCLs.

[1]We will not provide references for all sub-standards, but these can be easily found based on their names via IEEE Xplore.

Problem formulation: In this paper we consider real-time applications implemented using both the TT and AVB traffic types running on TSN-based distributed architectures. We assume that the network topology is given. The applications messages are modelled as TT and AVB messages. Furthermore, we assume that the traffic type of each message is given. We are interested to synthesize the routing and the GCLs for TT traffic, such that both TT and AVB traffic is schedulable.

4.1.1 Related Work and Contribution

There is a lot of research work on deriving schedule tables for tasks and messages [CS16]. For TTEthernet, researchers have proposed strategies for the scheduling of TT frames on network links [TSPS14], which take into account the lower priority Rate-Constrained (RC) traffic type of TTEthernet. However, although there are similarities between TSN and TTEthernet, they differ in some significant aspects: Messages in TTEthernet consist of a single frame, whereas TSN messages may consist of multiple frames. Furthermore, TTEthernet schedule tables are specified for individual TT frames, whereas TSN specifies schedules for the output port queues, not frames. Consequently, all frames sharing the same queue are affected by the associated GCL. As a result, the work on TTEthernet scheduling is not directly applicable to TSN. In addition, the transmission of RC frames and the corresponding timing analysis for WCDs differ significantly compared to AVB.

For the GCL synthesis problem in TSN, researchers have started to propose several approaches, based on, e.g., Satisfiability/Optimization Modulo Theories (SMT/OMT) [CSCS16] and metaheuristics [DN16]. Deciding the routing of traffic flows is also an important problem. The TSN standards proposes dynamic routing and reservation mechanisms, such as IEEE 802.1Qca and IEEE 802.1Qcc. Such dynamic routing is appropriate for non-critical traffic. However, real-time and safety-critical traffic uses static routes, decided at design time. Hence, researchers have also addressed the problem of determining the static routes for TT [Nay17] and AVB flows [Lau16]. Researchers have addressed also the joint routing and scheduling problem, proposing solutions based on Integer Linear Programming (ILP) [SDT^{+}17] and a List Scheduling-based heuristic [PTO18].

However, all of the approaches for routing or GCL synthesis presented previously have looked at TT traffic in isolation, completely ignoring the impact on AVB traffic. The work in [GP18] so far is the only one which addresses the GCL synthesis for mixed-criticality applications in TSN. As we will show in the experimental results section, ignoring AVB traffic results in routes and GCLs that are optimized for TT at the expense of AVB traffic, which leads to very large WCDs for AVB.

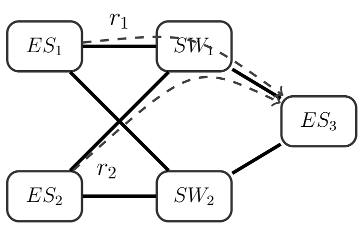
Contribution: In this paper, we propose a joint routing and scheduling of TT traffic in TSN taking into account the AVB traffic. To the best of our knowledge this is the first work dealing with the interdependence between AVB traffic and the TT routing and scheduling. Considering the effect of TT routing and scheduling on AVB traffic results optimized solutions guarantee the schedulability of TT traffic while at the same time reduces the WCDs of AVB traffic, such that its end-to-end latency requirements are fulfilled. To solve this problem we have developed a solution that integrates a K-Shortest Paths (KSP) heuristic for routing with a Greedy Randomized Adaptive Search Procedure (GRASP)-based metaheuristic for scheduling.

The paper is structured as follows: section 4.2 presents the architecture and applications models. We introduce briefly TSN and present how TT and AVB works in section 4.3. section 4.4 outlines our problem formulation and section 4.5 presents our proposed solution. The experimental results are in section 4.6 and the last section presents our conclusions.

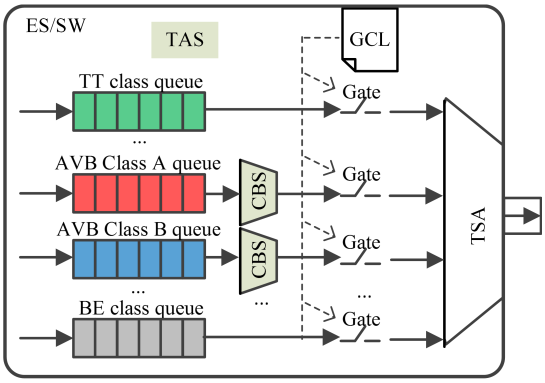
4.2 System Models

4.2.1 Architecture Model

The architecture model is an abstract representation of the physical TSN network, including end systems, switches, and physical links. The topology is modeled as an undirected graph $\mathcal{G}$, where the vertices represent devices in the network, i.e., end systems $\mathcal{ES}$, and network switches $\mathcal{SW}$, known in TSN also as bridges. The edges represent physical full-duplex links. A data link $dl_{i,j}$ is a directed communication link from a vertex v_i to another vertex v_j. 4.1a shows the topology of a network with three end systems, ES_1, ES_2, ES_3, and two switches, BR_1 and BR_2. A route r_i is a cycle-free ordered sequence of data links connecting one sending end system with one or more receiving end systems, via switches. Without loss of generality, we consider in this paper that the routes are unicast. 4.1a shows two routes: $r_1 = \{ES_1, BR_1, ES_3\}$, and $r_2 = \{ES_2, BR_1, ES_3\}$. The set of all routes in a network is denoted $\mathcal{R}$.



(a) Example architecture



(b) A TAS for an output port

Figure 4.1: TSN network and device architecture

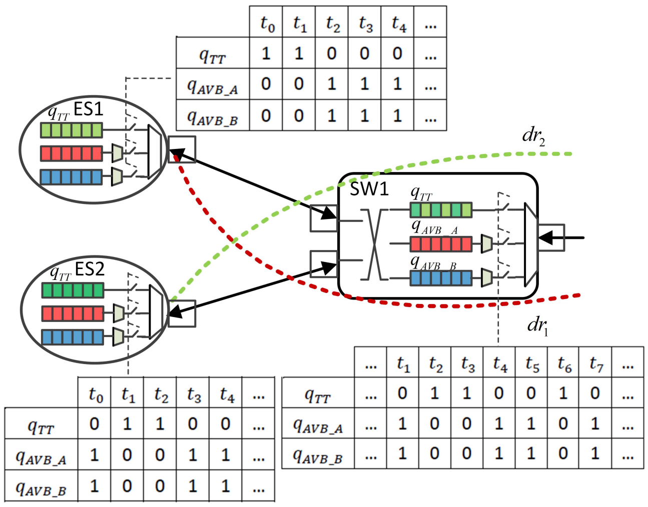
4.2.2 Application Model

The real-time applications are modeled as a set of messages which can be transmitted as TT or AVB flows. The set of flows in the system is denoted as $\mathcal{F} = \mathcal{F}^{TT} \cup \mathcal{F}^{AVB}$. Associated with each flow f_i is the tuple of attributes (v_s, v_t, T, D, P), where v_s denotes the sending end system and v_t denotes the receiving end system. The flows are periodic, with a period T and have a relative deadline D. P is the payload, or data size, of f_i. A single Ethernet frame transmits a payload of at most 1500 bytes (B), the so-called Maximum Transmission Unit (MTU). If the data size is larger than MTU, the message is fragmented into multiple frames, f_i^k denoting the k^{th} frame of the flow f_i. Table 4.1 shows eight sample flows, four TT flows f_1 to f_4 and four AVB flows f_5 to f_8.

A *routing* $R : \mathcal{F} \mapsto \mathcal{R} \cup \{\emptyset\}$ is a function which maps a flow to the route on which that message is forwarded. To show that a flow f_i has no assigned route we use the notation $R(f_i) = \emptyset$. In this work the routing R has to be decided. $U(R, dl_{i,j})$ denotes the *utilization* on link $dl_{i,j}$ for the routing R. It represents the sum of the bandwidth of the flows routed through the link $dl_{i,j}$ and is defined

Table 4.1: Example application model

flow	type	v_s	v_t	T (μs)	D (μs)	P (B)
$f_1 \cdots f_4$	TT	ES_1	ES_3	150	150	750
$f_5 \cdots f_8$	AVB	ES_2	ES_3	150	100	1500



	t_0	t_1	t_2	t_3	t_4	...
q_{TT}	1	1	0	0	0	...
q_{AVB_A}	0	0	1	1	1	...
q_{AVB_B}	0	0	1	1	1	...

	t_0	t_1	t_2	t_3	t_4	...
q_{TT}	0	1	1	0	0	...
q_{AVB_A}	1	0	0	1	1	...
q_{AVB_B}	1	0	0	1	1	...

	...	t_1	t_2	t_3	t_4	t_5	t_6	t_7	...
q_{TT}	...	0	1	1	0	0	1	0	...
q_{AVB_A}	...	1	0	0	1	1	0	1	...
q_{AVB_B}	...	1	0	0	1	1	0	1	...

Figure 4.2: TSN network with internal queues, gates and GCLs [ZPZL18]

as:

$$U(R, dl_{i,j}) = \sum_{f_k \in \mathcal{F} | dl_{i,j} \in R(f_k)} \frac{f_k.P}{f_k.T}.$$

For a route $r \in R$ the utilization U represents the maximum utilization of links composing the route, i.e., $U(R, r) = \max_{dl_i \in r} U(R, dl_i)$.

4.3 TSN Protocol

TSN is based on the switched multi-hop network architecture from IEEE 802.3 Ethernet. Switches interconnect end systems via *full-duplex* links, meaning that the physical links enable transmission in both directions simultaneously.

Ethernet frames contain IEEE 802.1Q headers, with two fields of importance to TT traffic:

- VLAN IDentifier (VID) is a 12-bit field specifying the Virtual LAN of a frame. This is used to distinguish frames from different messages.
- Priority Code Point (PCP) is a 3-bit field specifying the priority level, i.e., the traffic class such as TT, AVB, or BE. Furthermore, it defines which queue the frame is assigned to within a switch.

An Ethernet switch has ingress (incoming) and egress (outgoing) ports connecting it via links to surrounding switches and end systems. Each egress port typically has eight queues for storing frames that wait to be forwarded on the corresponding link, one or more TT queues, two for AVB (Class A and B respectively) and the remaining queues are used for BE. Figure 4.2 shows part of a TSN network.

4.3.1 TT Traffic

IEEE 802.1ASrev provides a clock synchronization protocol to obtain a global time base for TT transmission. Taking advantage of the global synchronized clock, IEEE 802.1Qbv defines a Time-Aware Shaper (TAS) to achieve low latency for TT traffic by establishing completely independent time windows by opening and closing the gates. Interference from lower priority traffic is prevented by closing the gates of the remaining queues, see 4.1b. When the egress port is idle, the next frame is selected for transmission from the queue with highest priority among the queues with open gates. Opening queues in a mutually-exclusive fashion, allows for full control of forwarded frame.

A Gate Control List (GCL) defines for each egress port, when the queue gates are open and closed. In Figure 4.2 they are depicted as tables. 1 and 0 in the GCL represent an open and closed gate, respectively. Using the GCLs to schedule forwarding of frames in a route from sender to receiver, enables very low latency and jitter for TT traffic, making it suitable for hard real-time communication.

The GCLs can be constructed in such a way that AVB and BE traffic are prevented from initiating transmission in time slots reserved for TT frames. However, nondeterminism could still occur due to interference with other TT flows. When a frame is scheduled for transmission on a link in a given time interval, the corresponding GCL is set to open the associated gate in that interval. Suppose something goes wrong, so the frame is not fully received, or is not the first frame in the queue as expected. Then the link transmits the wrong frame or remains idle when it should be transmitting. Consequently, nondeterminism is introduced, which means timeliness is compromised, see [CSCS16] for an in-depth discussion. Similar to the related work on GCL synthesis [CSCS16], we will determine the GCLs such that the non-determinism is avoided, see subsection 4.4.1 for a discussion.

Integration modes: When there are mixed-criticality frames within the same network, TT traffic might be delayed by AVB or BE traffic that is already being transmitted at the time of the schedule trigger for TT (i.e., gate open for the respective queue). In order to reduce this delay, TSN introduces two mechanisms. The first is referred to as *non-preemption* mode. The gate of lower-priority traffic can be closed in advance of the TT schedule event such that the port is available for the TT traffic. This mechanism is similar to the "guard band" approach found in TTEthernet [CS16]. The second mechanism is *preemption*, defined by IEEE 802.1Qbu, where the transmission of an AVB (or BE) frame will be interrupted by the transmission of a TT frame and resumed once the TT frame has been fully transmitted. As specified in IEEE 802.1Qbu even in the case of preemption mode the fragments of the preempted lower priority frames should be transmitted by well formatted Ethernet frames, i.e., the so-called smallest non-preemptable fragment consists of an Ethernet preamble, a minimum of 64 B of data and the trail which further consists on a CRC code and an inter frame gap [TE16]. Please note that for the first integration mode the delay of TT frames is 0 while for the second mode it is upper bounded by the transmission duration of the smallest non-preemptible fragment.

4.3.2 AVB Traffic

The availability of an AVB queue is also determined by a Credit-Based Shaper (CBS) and the purpose of CBS is to prevent the starvation of lower priority flows. Hence, an enqueued AVB frame is allowed to be transmitted if (i) the queue gate is open, (ii) the CBS allows it and (iii) there are no other higher priority AVB frames being transmitted.

The CBS standardized in IEEE 802.1Qat in conjunction with the amendments in IEEE 802.1Qbv makes the queue available for transmission whenever the amount of *credit* is positive or zero. The credit is initially zero, it is decreased with a *sending slope* (*sdSl*) while transmitting and frozen while the gate is closed. Transmission is only initiated when credit is non-negative. The credit is increased with an *idle slope* (*idSl*) when frames are waiting, but they are not being transmitted. If the queue is emptied while the credit is positive, the credit is reset to zero. The idle and sending slopes are configuration parameters described in IEEE 802.1Qbv; the idle slope is defined as fraction of link speed and the sending slope as difference between idle slope and link speed.

Using the example in Figure 4.3 we show how CBS works, considering also TT and BE traffic. Rectangles on the first timeline represent the transmission of frames and down arrows on top give the frames arrival times, for example $a^i_{AVB\_A}$ indicates the arrival time of the frame f_i. The lines on the timeline show the variation of credit for respective AVB class, where AVB Class A and B are respectively shown with red and blue. Figure 4.3a considers the non-preemption integration mode. An AVB Class A frame $f^1_{AVB\_A}$ arrives at t_0; meanwhile, a BE frame is transmitting. Due to the non-preemption of BE

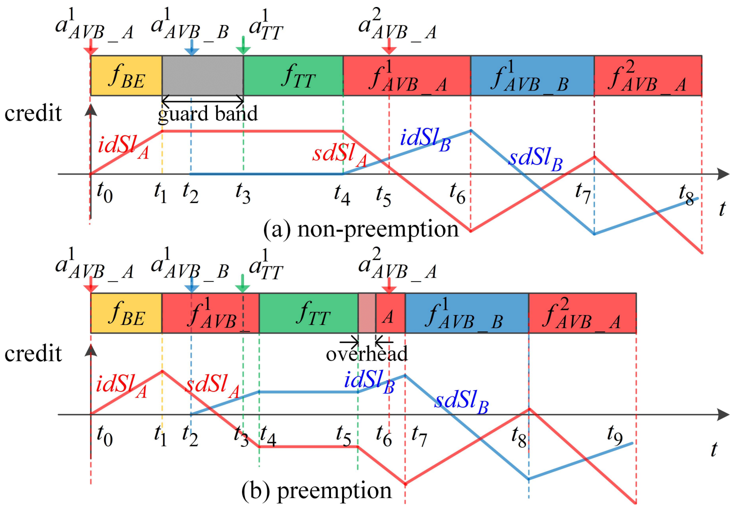


Figure 4.3: Example AVB transmission [ZPZL18]

frames, $f^1_{AVB\_A}$ has to wait until f_{BE} finishes its transmission, and credit A is increased with the idle slope $idSl_A$. At time t_1, the transmission of f_{BE} is completed. However, the AVB gates are closed due to the reservation for TT traffic and insufficient idle interval (caused by the guard band) for the whole frame $f^1_{AVB\_A}$ transmission. Therefore, credit A is frozen during $[t_1, t_4]$ when AVB gates are closed. When an AVB Class B frame $f^1_{AVB\_B}$ arrives at t_2, its credit is also frozen. From time t_4, since the gate for TT queue is closed and due to the higher priority of Class A, $f^1_{AVB\_A}$ is allowed to be transmitted. The credits for A and B are, respectively, decreased and increased with the sending slope $sdSl_A$ and idle slope $idSl_B$. During the transmission of $f^1_{AVB\_A}$, another frame $f^2_{AVB\_A}$ is enqueued in the Class A queue at time t_5. Then, at t_6 when frame $f^1_{AVB\_A}$ finishes, there are two frames $f^2_{AVB\_A}$ and $f^1_{AVB\_B}$ waiting to be transmitted. But credit A at this time is negative, therefore $f^2_{AVB\_A}$ is not allowed to be transmitted and $f^1_{AVB\_B}$ has the permission to be transmitted. At the end of $f^1_{AVB\_B}$ transmission, $f^2_{AVB\_A}$ starts its transmission as credit A has been increased to a non-negative value.

In Figure 4.3b, we present the preemption mode. The arrival times are the same as in Figure 4.3a. However, due to the preemption integration mode, the TT frame f_{TT} is delayed from t_3 to t_4, and when $f^1_{AVB\_A}$ resumes its transmission after being preempted, we have to consider an additional overhead (depicted).

4.4 Problem Formulation

The problem addressed in this paper is: Given a TSN network topology $\mathcal{G}$, and a set of TT and AVB flows $\mathcal{F} = \mathcal{F}^{TT} \cup \mathcal{F}^{AVB}$ determine an implementation ϕ such that: (i) all the flows are schedulable, i.e., the $WCD(f_i) \leq f_i.D$ for all TT and AVB flows. Determining ϕ means: (1) deciding the route $R(f_i)$ for each TT flow $f_i \in \mathcal{F}^{TT}$, (2) deciding the number of TT queues, (3) mapping the TT flows to egress port TT queues and (4) deriving the GCLs $\mathcal{GCL}$.

Note that mapping of AVB flows to AVB queues is decided by their class, i.e., AVB Class A flows are assigned to AVB Class A queue and B flows to Class B queue. Our proposed solution can also determine the AVB routes at the same time with the TT routes. However, the focus of this paper is on determining the routing and scheduling of TT flows, so we consider the AVB routing given and fixed. The routing of AVB flows aiming at reducing their WCDs has been addressed by us in [Lau16].

4.4.1 GCL Synthesis for TT

Let us consider the example from Figure 4.4 where we have the two flows from 4.4a routed as indicated in 4.4b.

Figure 4.4 shows GCLs using a Gantt chart, depicting how TT frames are transmitted. The x-axis represents time dimension, while y-axis is related to output ports of nodes. Moreover, the rectangles represent TT frames' transmission. The left side of rectangle is the start time of the transmitted frame, and its width represents the transmission duration which is related to the frame size and the physical link rate. To illustrate the queue usage, we use thin rows labeled q_i below the link schedules showing when frames are in the queues of the respective egress port.

The GCLs must satisfy the following constraints:

- *Link congestion.* A data link is limited by its hardware to only transmit a single frame at a time, i.e., frames on the same link cannot overlap in the time domain. This corresponds to the property that boxes on the same row of Figure 4.4 do not overlap. The link can be seen as a shared resource, that can only be occupied by a single frame at a time.
- *Flow transmission.* A switch cannot forward a frame until the entire frame has been buffered in the switch. This introduces a forwarding delay for

each hop from source to destination. Due to the small synchronization error of the clocks between devices, the exact time when the entire frame has been received in a particular switch is unknown. Consequently, the time for forwarding the frame on the next link should take into account the worst-case synchronization error, δ.

- *Bounded end-to-end latency.* All TT flows must arrive within their relative deadline, i.e., the end-to-end latency cannot exceed the deadline. End-to-end latency is defined as the time from the sender initiates transmission of the first frame and until the last frame arrives at the receiver.
- *Deterministic queues.* Analogously to the *link congestion* property, a queue can be considered a shared resource, which can only be occupied by frames from a single flow at a time. In Figure 4.4 this corresponds to the property that queue utilization boxes do not overlap in the time domain, if they belong to the same queue. In addition, there should be a δ-sized spacing between queue utilization boxes of frames arriving at different ingress ports, to account for the worst-case synchronization error.

Under these assumptions, the GCLs are conceptually equivalent to the schedule tables presented in Figure 4.4. 4.4c and 4.4d shows two feasible schedules. GCLs can be optimized according to several criteria, e.g., TT queue usage and TT end-to-end latency. In order to improve queue usage, the frames of f_1 and f_2 should be rearranged in such a way that they both share the same queue in $[SW_1, ES_3]$ without occupying the queue at the same time. 4.4c shows such a schedule, where they both use q_1. Notice that the queue utilization boxes do not overlap.

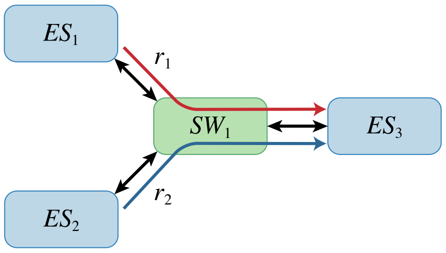
On the other hand, minimizing queue usage has a negative effect on end-to-end latency. In 4.4c the frames of f_2 have been spaced further apart, thereby increasing the end-to-end latency. Instead, the schedule could be optimized with respect to end-to-end latency as shown in 4.4d. In this schedule, two queues are used but the frames of f_2 are grouped closer together compared to 4.4c resulting in a smaller end-to-end latency. This example shows that a desirable schedule is a tradeoff between queue usage and end-to-end latency.

4.4.2 Routing for TT

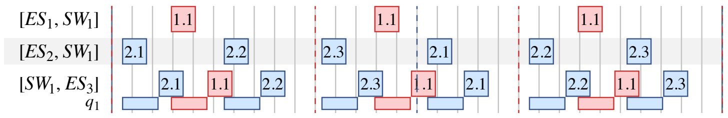
Let us illustrate the importance of optimizing the routing for TT flows. Let us consider the TT flows in 4.5a to be routed and scheduled in the architecture depicted in Figure 4.5, consisting of four end systems interconnected with five switches. If we use the shortest path routing, which would intuitively reduce

flow	v_s	v_t	r	$T(\mu s)$	$D(\mu s)$	$P(B)$
f_1	ES_1	ES_3	r_1	100	100	1500
f_2	ES_2	ES_3	r_2	150	150	4500

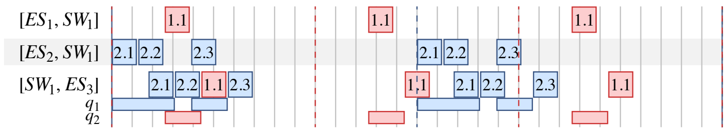
(a) Set of TT flows with attributes



(b) Network topology and routing of TT flows



(c) Minimum queue usage



(d) Minimum end-to-end latency

Figure 4.4: Example GCL synthesis for TT

the latency of TT flows, we obtain the two routes r_1 and r_2 as depicted in 4.5b. Flows f_1, f_2 and f_3 have the route r_1 and flows f_4, f_5 have the route r_2. In this situation, flow f_5 is not schedulable (does not meet its deadline), due to the congestion on the link $[SW_1, SW_2]$.

We can make the TT flows schedulable by rerouting one of the flows reducing thus the congestion on the link $[SW_1, SW_2]$. For example, flow f_1 can be routed on a longer route instead of the shortest path, e.g., $\{ES1$, $SW1$, $SW5$, $SW4$, $ES4\}$. The optimal routing, which minimizes the TT end-to-end latencies, is depicted in 4.5c (we label the routes with the flow numbers), which makes use of longer routes for several flows. This example shows that the routing of TT flows impacts their scheduling, and has to be considered at the same time with the GCL synthesis.

4.4.3 AVB-Aware TT Routing and Scheduling

So far, we have ignored the AVB flows. Let us illustrate the importance of taking into account the AVB flows during the TT routing and scheduling. For this example we are using the network topology from 4.1a, which has three end systems, ES_1 to ES_3, and two switches, BR_1 and BR_2, implementing the set of flows presented in Table 4.1 with four TT and four AVB flows, respectively. All AVB flows are Class A, with the default idle slope of 75%. Each flow is packed in one frame and all links have a speed of 1 Gbps, resulting in a transmission time C of 6.33 μs for TT frames and 13.24 μs for AVB frames. In this example we consider the non-preemption integration mode.

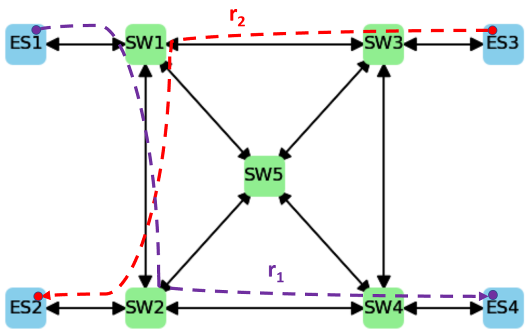
A flow is schedulable if the frames arrive before their deadlines at the destination, even in the worst-case scenario captured by $WCD(f_i)$. For TT, the WCDs are determined directly by the GCLs. However, the WCD of an AVB flow f is determined by its *worst-case scenario*, i.e., the situation that delays f the most. An AVB frame will be delayed by other AVB and TT frames (including the guard band of those TT frames) sharing the same output port.

In the following examples we are going to show the schedule tables for TT flows and we will illustrate the worst-case scenario for one AVB flow, namely f_5.

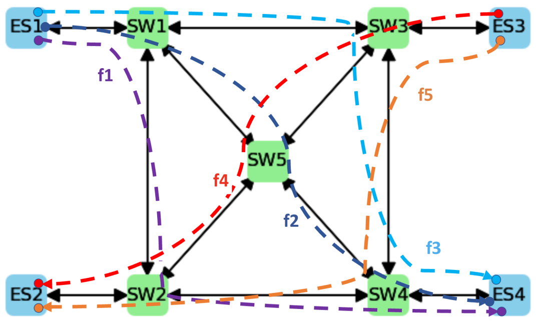
See section 4.3 for how to read the schedule. In addition, the number on top of rectangle box represents the transmission time of the frame, and the number on top of a blank interval is the waiting time due to the negative credit of CBS or for timely block reasons. The Δ in the figure means that the frame arrives just at the instant when the remaining time is slightly less than its transmission time. Similar to Figure 4.3, we use downward pointing arrows to show the

Flows	Endpoints (v_s, v_t)	$P(B)$	$T(\mu s)$
f_1, f_2, f_3	$ES1 \Rightarrow ES4$	1500	100
f_4, f_5	$ES3 \Rightarrow ES2$	1500	100

(a) Flows example for routing



(b) Shortest path routing

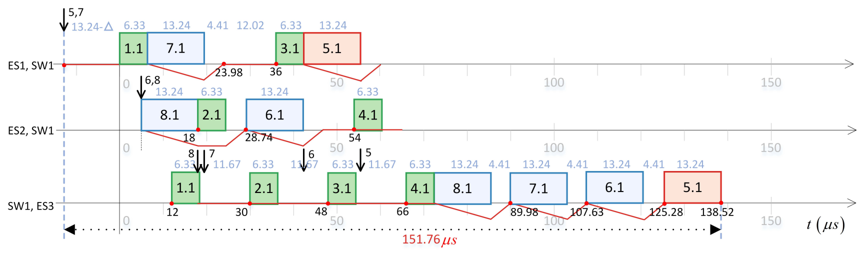


(c) Optimized routing

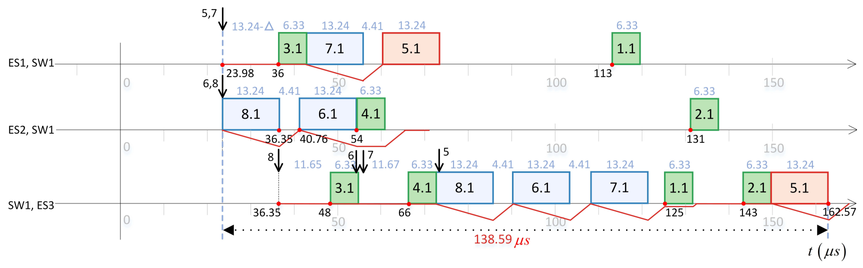
Figure 4.5: Example routing optimization for TT

arrival times for AVB frames that create the worst-case scenario for f_5. Note that the GCLs are cyclic and in the worst case f_5 may arrive in the previous hyperperiod and it cannot start transmitting due to the timely block. The AVB credit for the queue of f_5 is depicted below the timelines using a red line.

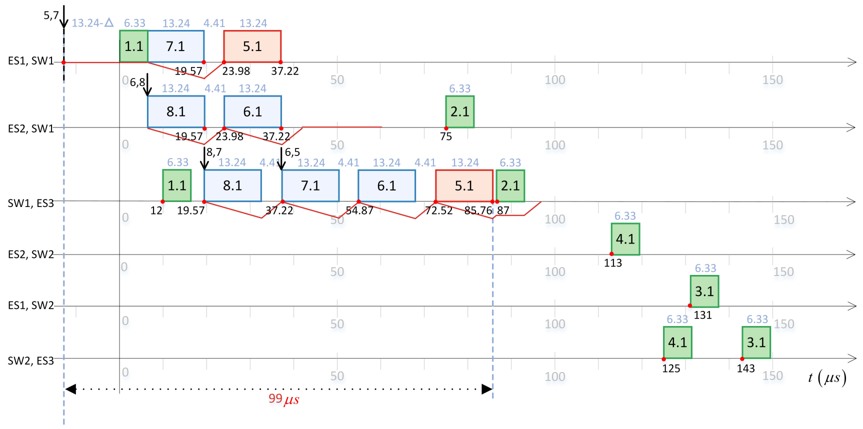
4.6a shows the optimal schedule for TT flows, which minimizes their WCDs by scheduling them as soon as possible. Only one queue is used for TT and all AVB flows are in the AVB Class A queue. As expected, the TT flows are schedulable, but all AVB flows have a WCD of 151.76 $\mu s > D$ so they are not schedulable. The worst-case scenario resulting in the WCD of 151.76 μs for f_5 is depicted in 4.6a.



(a) Optimal TT schedule: AVB flows are unschedulable; WCD for f_5 is 151.76 μs



(b) AVB-aware TT schedule: WCDs of AVB flows are decreased to 138.59 μs



(c) Rerouting of some TT flows: *all* AVB flows are *schedulable*; WCDs for AVB flows are 99 μs

Figure 4.6: Motivational examples for considering AVB during TT routing and scheduling

However, if we construct an optimized GCL, as depicted in 4.6b, we can decrease the WCDs of all AVB flows to 138.59 μs. In this example, we have rescheduled the TT flows f_1 and f_2 by delaying them into the second half of the hyperperiod.

This keeps the TT flows schedulable (still preserving their WCDs), but has the benefit of creating space for the AVB flows, decreasing their delay even in their worst-case. We illustrate the reduction of the WCD of AVB frame f_5 in 4.6b compared to 4.6a.

Another choice is to reroute the TT flows f_3 and f_4 through switch BR_2 as depicted in 4.6c. As we can see, this preserves the schedulability of TT flows, as in the previous examples, but now also all AVB flows are schedulable. By rerouting the TT traffic, the WCD of AVB flow f_5 becomes 99 $\mu s \leq D$.

The examples have shown the importance of carefully deciding the routes and GCLs for the TT traffic such that both TT and AVB are schedulable. Ignoring the AVB traffic when deciding on the routing and scheduling of TT, as is the case with all the related work, results in large AVB WCDs that miss the deadlines.

Algorithm 4.1 $JRS(\mathcal{G}, \mathcal{F})$

```
1: $\phi \leftarrow \emptyset$
2: repeat
3:     $R \leftarrow RoutingHeuristic(\mathcal{G}, \mathcal{F})$
4:     $\phi' \leftarrow SchedulingMetaheuristic(\mathcal{G}, \mathcal{F}^{TT}, R)$
5:     if $Obj(\phi', \mathcal{F}^{AVB}) < Obj(\phi, \mathcal{F}^{AVB})$ then
6:         $\phi \leftarrow \phi'$
7:     end if
8: until stopping criterion not met
9: return $\phi$
```

4.5 Optimization Strategy

The problem presented in section 4.4 is NP-hard. Exhaustively just enumerating every path between two vertices has been proven NP-hard [Val79]. Also, the corresponding decision problem of the TT scheduling problem, namely the flow-shop scheduling, is NP-complete [GJS76]. Thus for a joint scheduling and routing problem to exhaustively evaluate every schedule and routing combination leads to an intractable amount of combinations that have to be evaluated. To solve our network design problem we use an integrated heuristic-metaheuristic strategy, i.e., a routing heuristic based on K-Shortest Path (KSP) method [Yen71] and a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic [RR14] for the scheduling. The integrated strategy is further called Joint Routing and Scheduling *JRS* and is presented in Algorithm 4.1.

Heuristics are not guaranteed to find the optimal solution; successful heuristics are able to find good quality solutions to large problem sizes in a reasonable time. Heuristics are algorithms specialized for a particular problem (in our case, routing), whereas metaheuristics are more general optimization algorithms that can be applied to a wide range of problems (we use the GRASP metaheuristic for scheduling).

JRS takes as an input the architecture $\mathcal{G}$ and the set of flows $\mathcal{F}$ and produces at the output a routing and scheduling solution ϕ. *JRS* generates multiple routing and scheduling alternatives, further called solutions (lines 3, 4) and evaluates each solution (line 5) attempting to find that solution which minimizes the *objective function.* The objective function is presented in subsection 4.5.1. Our proposed routing and scheduling algorithms are presented in sections 4.5.2 and 4.5.3, respectively. *JRS* terminates after a given time limit is reached or if there is no improvement in the objective function after a given number of iterations.

4.5.1 Objective Function

We are interested to find solutions that meet the deadlines for both TT and AVB flows. We consider a solution to be invalid if the TT deadlines are not satisfied. On each iteration of JRS and during the local search, the quality of a valid solution x is evaluated using an objective function $Obj(x, \mathcal{F}^{AVB})$ that checks if the AVB flows are schedulable, driving thus the search towards schedulable solutions:

$$Obj(x, \mathcal{F}^{AVB}) = \sum_{f_i \in \mathcal{F}^{avb}} \max(0, WCD(f_i) - f_i.D). \tag{4.1}$$

The objective function captures the sum of *tardiness* for each AVB flow, i.e., the time it takes to arrive at its destination after its deadline has passed.

To validate and evaluate a solution we need to compute the WCDs for both TT and AVB flows. Due to the deterministic behavior of TT traffic the WCD for a TT flow is easily computed from the schedule table as the difference between the time when the last frame arrives at the receiver and the time when the sender initiates transmission of the first frame. However, to determine the WCDs of

AVB flows in presence of TT traffic, we have to use a schedulability analysis for AVB. Although such analyses have been proposed in the past, they have considered AVB in isolation, ignoring TT traffic. Only recently researchers have proposed AVB analysis to consider the influence of GCLs. Thus, [Lau16] has extended the *AVB Latency Math* from IEEE 802.1BA, but this approach is very pessimistic. In this paper, we use the AVB schedulability analysis from [ZPZL18] that: (i) is based on Network Calculus, (ii) takes into account the GCLs, (iii) supports both AVB Class A and B and (iv) considers all integration modes, considerably reducing the pessimism compared to [Lau16].

4.5.2 Routing Strategy

Our routing heuristic, presented in Algorithm 4.2, takes as input the architecture $\mathcal{G}$ and the set of TT flows $\mathcal{F}^{TT}$, including for each flow $f_i \in \mathcal{F}^{TT}$ the sending $f_i.v_s$ and receiving $f_i.v_t$ end systems. The strategy outputs the routing R which maps a route to each flow.

Initially the routing R is empty, which means that routes have to be found for all TT flows. As researchers demonstrate [SDT$^+$17][Lau16], when we target time-sensitive applications, the shortest path routing may not lead to the smallest WCDs. However, enumerating all possible cycle-free paths (the full search space) to find the optimal routes is intractable. Hence, our strategy is to reduce the search space by using the K-Shortest Path (KSP) algorithm [Yen71], which generates K unique routes of increasing length (the set R^{KSP}), starting from the shortest route. Our idea is that good quality routing solutions can be found by combining routes which, although are not the shortest routes, they are not excessively long. K is a parameter that controls how many routes are generated. The K parameter is randomly picked from the interval $[1, \overline{K}]$ (line 3), where $\overline{K}$ represents the upper bound of K, see [ZP18] for details on how this upper bound can be determined experimentally for each flow based on the size of the topology.

The routing heuristic iteratively selects a route for each TT flow attempting to evenly distribute the link utilization. The idea is that a high link utilization increases the WCDs. Thus, to determine a route $R(f_i)$ for a flow $f_i \in \mathcal{F}^{TT}$ the routing heuristic: (1) generates the reduced set of routes R^{KSP} line 4 and (2) selects the *least utilized* route from the generated set and adds it to the partial routing R (line 5). The utilization $U(R, R(f_i))$ of a route $R(f_i)$ from a partial routing solution R represents the maximum link utilization of the links composing the route, see subsection 4.2.2 for how we calculate the utilization.

If there are multiple routes with the minimum utilization, the heuristic selects a route randomly. By choosing randomly K and the least utilized route we are able to diversify the routing solutions output by our *RoutingHeuristic*, which helps *JRS* in Algorithm 4.1 to explore the solution space.

Algorithm 4.2 *RoutingHeuristic*$(\mathcal{G}, \mathcal{F})$

```
1: $R \leftarrow \emptyset$
2: for $f_i \in \mathcal{F}^{TT}$ do
3:     $K \leftarrow PickK(f_i)$
4:     $R^{KSP} \leftarrow GenerateRoutes(\mathcal{G}, f_i.v_s, f_i.v_t, K)$
5:     $R \leftarrow R \cup \{SelectRoute(R, f_i, R^{KSP})\}$
6: end for
7: return $R$
```

4.5.3 Scheduling Strategy

Algorithm 4.3 *SchedulingMetaheuristic*$(\mathcal{G}, \mathcal{F}, R, \gamma, \pi)$

```
1: $x \leftarrow \emptyset$
2: repeat
3:     $x' \leftarrow GreedyRandomized(\mathcal{G}, \mathcal{F}^{TT}, R, \gamma)$
4:     $x' \leftarrow LocalSearch(x', \mathcal{G}, \mathcal{F}^{TT}, R, \pi)$
5:     if $Obj(x', \mathcal{F}^{AVB}) < Obj(x, \mathcal{F}^{AVB})$ then
6:         $x \leftarrow x'$
7:     end if
8: until termination criteria not met
9: return $x$
```

GRASP is well-suited for combinatorial optimization problems, where an initial solution can be efficiently constructed in a greedy manner. Each iteration of GRASP consists of two phases: (1) A *construction phase*, where an initial feasible solution is built, and (2) a *search phase*, where a neighborhood around the initial solution is examined for improving solutions. The construction phase contributes with diversification, and the local search with intensification, enabling convergence towards a global optimum.

Algorithm 4.3 presents our GRASP-based scheduling metaheuristic. As input, it takes (1) the architecture $\mathcal{G}$, (2) the set of flows $\mathcal{F} = \mathcal{F}^{TT} \cup \mathcal{F}^{AVB}$, (3) the routing R and (4) two parameters, γ and π, related to the construction and local search phases, respectively. Algorithm 4.3 outputs a feasible schedule x, which is the best scheduling solution found throughout the search in terms of the objective function from subsection 4.5.1. Initially, x is empty (line 1), indicating that a feasible solution is yet to been found, i.e., the set of scheduled flows TT is empty.

In each iteration, a new scheduling solution x' is generated in a greedy randomized fashion (line 3) using a constructive scheduling heuristic (*GreedyRandomized*, subsubsection 4.5.3.1). *GreedyRandomized* contains a random element to ensure that different parts of the solutions space are explored in each iteration. The parameter γ defines the level of randomness. Too much randomness affects the quality of the initial schedules, whereas too little randomness affects diversification.

The initial solution is subsequently optimized via a local neighborhood search until reaching a local optimum (line 4). The local search destroys and repairs the current schedule to obtain new schedules. The parameter π specifies how much to destroy/repair in each iteration of the local search. If a new solution results in a better solution than the current best known, then the best solution is updated (lines 5 and 6). This repeats until a given execution time limit has been reached (termination criteria). The local search phase is described in subsubsection 4.5.3.3.

4.5.3.1 Greedy Randomized Heuristic

Algorithm 4.4 *GreedyRandomized*$(\mathcal{G}, \mathcal{F}^{TT}, R, \gamma)$

```
1: $x \leftarrow \emptyset$
2: $\mathcal{F}' \leftarrow SortByPeriod(\mathcal{F}^{TT}, R)$
3: for $f_i \in \mathcal{F}'$ do
4:     $RCL \leftarrow RestrictedCandidateList(\gamma, f_i)$
5:     while $ScheduleFlow(x, f_i, RCL) = $ true do
6:         $x' \leftarrow x \cup \{f_i\}$
7:         $RCL.AddCandidate(x')$
8:     end while
9:     if $RCL.Length() > 0$ then
10:        $x \leftarrow RCL.GetRandomCandidate()$
11:    end if
12: end for
13: return $x$
```

The construction phase of the GRASP-based *SchedulingMetaheuristic* is presented in Algorithm 4.4. *GreedyRandomized* is a polynomial-time greedy randomized heuristic designed to find feasible schedules which serve as good starting points for the subsequent local search. Hence, it should be computed efficiently,

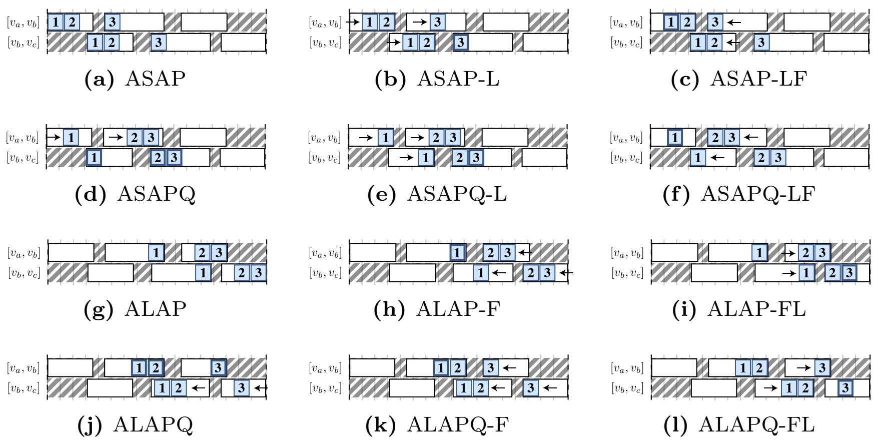


Figure 4.7: Heuristic variations for scheduling frames of a flow in an existing schedule

should not produce the same schedule in each iteration, and should not make obvious suboptimal decisions which the local search has to spend much time rectifying.

Flows are ordered by their period (line 2). To break ties, the route length is used as an indicator of how difficult flows are to schedule. Flows are scheduled one at a time in this order (lines 3-12). Given the current scheduling solution x, we attempt to schedule each unscheduled TT flow using several heuristic variations based on List Scheduling, see subsubsection 4.5.3.2. The Restricted Candidate List (RCL) (line 4) is a data structure that keeps track of the γ best schedules produced by the heuristics with respect to the objective function Obj. When all heuristics have been considered, a random schedule is chosen from among those in RCL (line 10).

It may be the case that the heuristics are unable to schedule a particular flow. In that case, no candidate is added to RCL (lines 5-8). If all heuristics fail, RCL is empty (lines 9-11), i.e., that particular flow is not included in the schedule, making it invalid.

TT flows are scheduled individually using *ScheduleFlow*, presented in the next section.

4.5.3.2 Schedule Flow

Given an existing partial scheduling solution x, several feasible schedules exist for a flow f. In this section we present a List Scheduling-based heuristic approach called *ScheduleFlow* for scheduling the individual frames of a single flow, while minimizing queue usage. The achieved schedule can then, in turn, be post-processed to minimize end-to-end latency. The heuristic strategy is generalized into multiple variations denoted $\mathcal{H}$.

ScheduleFlow schedules the frames of f sequentially, scheduling each frame on all links before moving on to the next frame. It continues in this way until either all frames in f are successfully scheduled, or until failing to schedule a particular frame, i.e., failing to determine offsets for the frame such that all constraints of subsection 4.4.1 are satisfied.

4.7a illustrates an "As Soon As Possible" (ASAP) approach to scheduling frames in a schedule. A frame is scheduled on its route, in-order, at the earliest possible offset where the link is idle and the queue is empty. If the frame is not assigned to the same empty-queue block as it was on the previous link, the algorithm backtracks and reschedules the previous frame to the next empty-queue block. *ScheduleFlow* succeeds if the last frame is scheduled within the deadline and fails otherwise. Analogous to ASAP, an "As Late As Possible" (ALAP) approach can be formulated by traversing the frames in reverse order, as well as scheduling on links in reverse order. The reader is referred to [RP17], for more details about determining feasible frame offsets using the ASAP and ALAP approaches.

Reducing queue usage: To minimize queue usage, the heuristic initially assigns all flow instances to the first queue. If the heuristic at some point fails to schedule a particular frame on one of its links, it may be because the queue assignment imposes too many restrictions on the feasible frame offsets. In this case, the queue assignment is incremented for some link on the route, before restarting the algorithm from the first frame. The idle-link and empty-queue blocks are used to determine which link to increment. ASAP heuristic increments the first queue assignment which allows a frame to start earlier than with the current queue assignment. When the algorithm terminates one of two things has happened: Either all frames have been scheduled, or some switch has no more queues available, i.e., the heuristic failed to schedule the flow.

Reducing end-to-end latency: Once a feasible solution has been found it can be post-processed to minimize end-to-end latency. Recall, that the end-to-end latency is the time from the offset of the first frame on the first link and until the finish time of the last frame on the last link. Hence, shifting the first frame to the right, or the last frame to the left reduces end-to-end latency. The intervals in which frames can safely be shifted without violating feasibility are computed from the empty-queue and idle-link intervals. A frame can be post-processed immediately when it has been scheduled on all links, or all frames can be post-processed together when the entire flow has been scheduled.

Figure 4.7 shows heuristic variations including the original ASAP heuristic (4.7a). White boxes represent the intervals where frames can be shifted. The main variation, ASAPQ, is illustrated in 4.7d. It reduces the time frames spend in queues by shifting each frame instance as close as possible to the frame instance on the next link. Frame instances on the last link are not moved, which is illustrated with a thick border in 4.7d. As an important side effect, the method reduces the overall time the queue is occupied, which could lead to better queue utilization and a lower total number of queues.

The remaining ASAP variations are different ways of post-processing the offsets once all frames have been scheduled by either ASAP or ASAPQ. In ASAP-L (4.7b) all frame instances are shifted toward the last frame instance to reduce end-to-end latency. The schedule produced by ASAP-LF (4.7c) has been through an additional post-processing step, where all frames instances are shifted toward the first frame instance. ASAPQ-L and ASAPQ-LF are variations of ASAPQ that have been post-processed in the same two ways.

The same variations can be formulated for the ALAP heuristic, but every shift is reversed compared to ASAP. Consequently, the post-processing steps first move toward the *first* frame instance, then the *last*. 4.7g– 4.7l depict the variations for ALAP. In total, twelve heuristic variants are used, targeting both the latency and the queue usage. We refer the reader to [RP17] for more details.

4.5.3.3 Local Search

The purpose of the local search phase is to intensify the search by investigating a well-defined neighborhood of solutions similar to the current solution. This corresponds to schedules where the majority of TT flows are scheduled exactly as in the current solution. It is likely that a better solution arises from rescheduling only a couple of TT flows. The local search attempts to identify such rearrangements by removing a small subset of TT flows, and rescheduling them in a different way.

The destroy and repair mechanisms of the local search rearranges flows compared to the original static order given by *SortByPeriod*. Thus, the local search can recover from a suboptimal ordering of flows in the construction phase.

The neighborhood is defined as follows: All the schedules which can be constructed by removing up to π flows from x, and subsequently rescheduling them using one of the scheduling heuristics. If a new, improving solution is discovered, the neighborhood search is repeated for the new solution. The local search continues in this way until reaching a local minimum, from which no solution from the neighborhood improves the current solution, or until exceeding the time limit.

4.6 Experimental Evaluation

We have performed three sets of experiments. In the first two sets of experiments we focus on the ability of our approach to determine good quality solutions for the TT routing and scheduling in a reasonable time. We are interested if our Joint Routing and Scheduling (JRS) approach scales well with large problem sizes. For these experiments we ignore AVB. Hence, in the first set of experiments (subsection 4.6.1) we evaluate the quality of our GRASP-based Scheduling Heuristic for TT, and in the second set of experiments we evaluate the importance of considering the TT routing during the GCL synthesis (subsection 4.6.2). In the last set of experiments we take AVB into account and evaluate JRS in terms of its ability to determine schedulable solutions for both TT and AVB.

4.6.1 Evaluation of GRASP-based Scheduling Heuristic for TT GCL synthesis

In the first set of experiments we were interested to evaluate the quality of our GRASP-based GCL synthesis approach. We have used the GRASP implementation from subsection 4.5.3 and we use two objective functions for GRASP, the normalized queue usage, denoted k_N, and the normalized end-to-end latency, denoted Λ_N. $\mathrm{k}_N(x)$ is a mapping of queue usage for TT traffic to the interval $[0;1]$ as shown in Equation 4.2.

$$k_N(x) = \frac{k(x) - \underline{k}(x)}{\overline{k}(x) - \underline{k}(x)} \tag{4.2}$$

where $k(x)$ denotes the total number of queues used for TT traffic across all egress ports. $\underline{k}$(x) and $\overline{k}$(x) are lower and upper bounds, respectively. The lower bound is assuming a single TT queue in all egress ports forwarding TT traffic, and the upper bound assumes the minimum of the available queues in the egress port and the number of TT flows forwarded through that egress port.

The total end-to-end latency is normalized in Λ_N in a similar way. The lower bound assumes that all flows are scheduled independently, i.e., without interference from other TT flows. The upper bound assumes the worst-case scenario where all flows have end-to-end latencies equal to their deadline.

To evaluate GRASP, we have implemented the OMT approach from [CSCS16] and the ILP approach from [PRCS16], which both minimize the number of queues (k). The values of k are presented in the table, including the lower and upper bounds, and the normalized value. We have compared the three approaches on the test cases from [PRCS16], and our GRASP has been able to obtain the same optimal solutions in less than 1 second for all test cases as shown in Table 4.2.

Further, we considered six topologies of varying size. The topologies are industrial sized, and are derived from the work presented in [SCS14]. The topologies are grouped into three categories based on their size. There are three small topologies (G1, G2 and G3, with up to 4 ESes and 3 NSes), two medium (G4 and G5, with up to 48 ESes and 28 NSes), and one large (G6, with 256 ESes and 146 NSes), see Table 4.4 for the number of flows on each topology size. The network precision is assumed to be $\delta = 5.008\ \mu s$. The transmission rate for all links is fixed at 1 Gbps, and the propagation delay of each link is assumed negligible, i.e., it is set to zero. Every egress port has eight queues.

The hyperperiod of all flows defines the width of the schedule, and has a major impact on the complexity of the problem. Thus, the hyperperiod is an important aspect to consider, when evaluating performance. We define three hyperperiods of 1 ms, 6 ms, and 30 ms. For each choice of hyperperiod we define a set of short periods and a set of long periods as presented in Table 4.3. Short-period flows have a data size of either one, two, or three times the MTU of 1500 bytes. Long-period flows have data sizes 10, 20, 40, 60, or 100 times MTU. The choice of periods and data sizes are inspired by [CSCS16].

Table 4.2: Comparison of ILP, OMT, and GRASP

	running time (s)			*queue usage*			
ID	ILP	OMT	GRASP	k	$\underline{k}$	$\overline{k}$	k_N
T01	0.66	0.81	0.32	2	2	5	0
T04	2.49	2.46	0.21	2	2	5	0
T05	3.73	3.43	0.34	2	2	3	0
T10	4.70	5.12	0.72	4	4	8	0
T11	16.54	12.94	0.84	3	3	7	0
T12	210.03	34.33	0.69	5	5	9	0
T14	39.06	22.87	0.84	2	2	3	0
T18	10.98	7.17	0.56	2	2	5	0

In order to generate flows, that yield difficult scheduling problems in terms of queue usage and end-to-end latencies, the link utilization should be relatively high. Hence, synthetic applications are generated by repeatedly adding short-period and long-period flows to the set of flows. The sending and receiving end systems are randomly chosen among the end systems in the topology. This procedure is repeated until multi-queue scenarios arise.

For each choice of topology and hyperperiod, we generate 30 test cases with high link utilization. In total we use 540 test cases, 90 for each of the six topologies.

Table 4.4 shows the average number of flows and frames for every pair of topology class and hyperperiod. Overall, the test instances range from a few hundred frames to tens of thousands of frames.

We have extended the ILP formulation presented in [PRCS16], which minimizes queue usage, to also feature end-to-end latency minimization. For the ILP formulation, the Gurobi [Gur] solver was given a time limit of 4 hours, after which it returns the best feasible solution. The ILP approach is intractable for many of the test instances, especially for larger hyperperiods. The results are compared with GRASP in 4.8a and 4.8b for the subset of test cases solved by ILP (the x-axis shows the topologies G1 to G6, each with flows of varying hyper-periods, 1 to 30 ms). Some data points are missing, because the ILP approach was unable to find feasible solutions within the time limit.

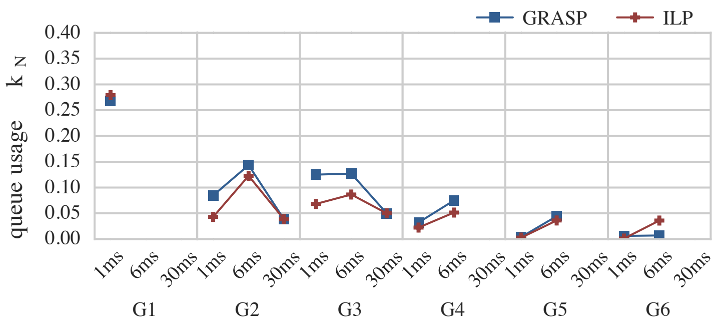
4.8a shows on the y-axis the normalized queue usage k_N, and 4.8b shows in μs the normalized end-to-end latency Λ_N. The ILP approach was able to solve 48% of the instances when minimizing queue usage and 42% when minimizing latency. On average, the ILP approach produced schedules with 17% lower queue usage in 4.8a and 51% lower end-to-end latency in 4.8b, but had a 15–20 times longer execution time.

Table 4.3: Combinations of periods in test cases

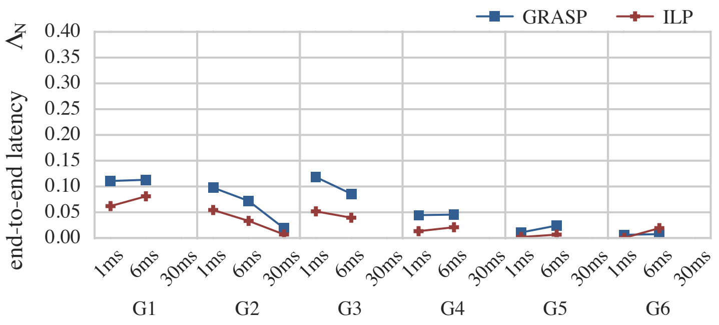
hyperperiod	short periods (μs)	long periods (ms)
1 ms	100, 200, 500	1
6 ms	100, 150, 200, 500	1, 2, 6
30 ms	100, 150, 200, 300, 500	5, 10, 30

Table 4.4: Average number of (flows, frames) of test cases

	topology size		
hyperperiod	small	medium	large
1 ms	$(17, 174)$	$(61, 548)$	$(358, 2078)$
6 ms	$(15, 436)$	$(55, 1193)$	$(254, 3682)$
30 ms	$(18, 737)$	$(63, 2944)$	$(327, 15167)$



(a) Average queue usage for GRASP and ILP



(b) Average end-to-end latency for GRASP and ILP

Figure 4.8: Comparison of GRASP and ILP

GRASP is able to significantly improve execution time compared to the ILP approach which is intractable for large instances, and is able to produce better schedules than a pure heuristic approach (e.g., variants of ASAP and ALAP). Its ability to minimize the objectives could be improved by increasing the time limit. Conversely, the time limit can be decreased in order to compute feasible schedules quickly. This flexibility makes GRASP well-suited to be used for GCL synthesis, where the schedules must take into account AVB flows.

Table 4.5: Comparison of shortest path routing with our proposed TT *RoutingHeuristic*

	TC1			TC2			TC3			TC4			Orion			SAE		
	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H	L	M	H
Shortest path	100	64	62	100	77	67	100	100	40	100	100	60	100	84	73	100	62	45
RoutingHeuristic	100	100	87	100	100	99	100	100	60	100	100	65	100	98	87	100	89	81

4.6.2 Evaluation of TT routing heuristic

In this section we are interested to evaluate the ability of our *RoutingHeuristic* from Algorithm 4.2 to find TT routes that improve the schedulability of TT frames. We have ignored AVB flows in these experiments and focused only on TT.

We have used four synthetic test cases, TC1–TC4 and two real-life test cases, "Orion" and "SAE". We have varied the size and type of network topology, i.e., topologies such as "mesh" that have more alternative routes connecting ESes and topologies with less alternative routes. Thus, TC1 is a mesh topology with 6 end systems and 8 network switches and TC2 is a mesh topology with 12 ESes and 14 NSes. TC3 and TC4 use a "ring" topology that has fewer alternative routes. In TC3, we have added more dataflow links such that TC3 is in-between a mesh and a ring topology. Both TC3 and TC4 have 12 ESes and 12 NSes. Finally, "Orion" and "SAE" are real-life topologies adapted from [TSPS14].

The results are presented in Table 4.5, where we have used the results obtained considering "Shortest path" routes as a baseline. In the table we show three setups for each test case: low utilization (L), medium utilization (M) and high utilization (H). The increasingly higher utilization has been obtained by increasing the number of TT flows; the TT flows were generated as presented in subsection 4.6.1. For each algorithm and test case, we show the percentage of TT flows that are schedulable (out of 100%).

As we can see from Table 4.5, as the size of the topology and the utilization increase, the shortest path routing has difficulties in finding schedulable solutions for the TT flows. Note that, for low utilizations it is easy to find schedulable solutions even without optimizing the routing. However, our *RoutingHeuristic* is able to significantly improve on the shortest path routing as the topologies get larger and more utilized.

As we can see from the results, for topologies that have alternative routes, optimizing routing is very important if we want to obtain solutions where TT flows can be scheduled, and our *RoutingHeuristic* is able to find such results in the short time limits imposed. We have used time limits of 1, 5, 15 and 30 minutes, corresponding to the topology size and utilization (the larger the topology and utilization, the longer the time limit). As expected, for topologies that do not have multiple alternative routes between end systems (e.g., TC3 and TC4), optimizing routing is less important.

Table 4.6: Experimental results for *JRS*

Test case	TT flows	AVB flows	End systems	Switches	TT latency		TT Queues		TT+AVB		*JRS*	
					Exec.	Sched.	Exec.	Sched.	Exec.	Sched.	Exec.	Sched.
Motiv.	4	4	3	1	1.1	No	0.8	No	32	Yes	21.9	Yes
TC5	12	4	5	2	2.2	No	2.6	No	403.9	Yes	148.3	Yes
TC6	35	26	32	18	8.7	No	9.6	No	612.8	Yes	527.7	Yes
TC7	427	464	256	146	708.6	No	628.8	No	728.5	No	534.6	Yes
Auto.	14	34	20	20	2.8	No	2.7	No	422.8	No	300.03	Yes

4.6.3 Evaluation of AVB-aware routing and scheduling

Our proposed TT routing and scheduling algorithms are the only approaches proposed so far in the literature that can take into account the AVB flows. Our AVB-aware Joint Routing and Scheduling (*JRS*) for TT flows is evaluated in this section. We have considered the non-preemption integration mode in these experiments, but our approach is able to handle all integration modes.

To evaluate *JRS*, we used five test cases, "Motiv." (the motivational example in Figure 4.6), three synthetic test cases, TC5–7 and an automotive test case "Auto.". For TC5–7 we used star and snowflake topologies, gradually increasing the size of the network. For "Auto", we used traffic flows obtained from an automotive manufacturer, implementing ADAS functions, and we have used the approach from [GZPS17] to generate the network topology. The details of the test cases, i.e., the number of TT and AVB flows and the number of ESes and NSes are in Table 4.6 columns 2–5.

We have run our proposed *JRS* optimization strategy on these test cases, and the results are presented in Table 4.6 under the heading *JRS*. We have compared our approach with three other approachs. (1) Thus, "TT+AVB" does not attempt to optimize routing (considers shortest paths) and instead uses the *SchedulingMetaheuristic* from Algorithm 4.3 to determine GCLs such that both TT and AVB flows are schedulable (i.e., it takes into account the AVB flows).

The two other approaches ignore AVB flows and also use shortest path routing. Thus, (2) in the "TT latency" aproach, where we minimized the latency of TT flows and (3) in the "TT queues" aproach we minimized the number of TT queues (in the hope of helping indirectly the AVB flows by reducing the number of higher-priority TT queues). These two approaches were also implemented using *SchedulingMetaheuristic*, but the objective function has been changed to minimize the TT latency and the number of TT queues, respectively, see subsection 4.6.1. The "Exec." columns show the algorithms' runtime in seconds. The TT flows were schedulable for all experiments. In the columns labeled "Sched." we show if the AVB flows were also schedulable.

As we can see from the table, if AVB is ignored during scheduling as is the case with the “TT latency” and “TT queues” approaches, we are not able to find schedulable solutions for the AVB flows. However, if we take into account the AVB flows during the scheduling (the “TT+AVB” heading in Table 4.6), we are able to find solutions where both TT and AVB flows are schedulable. However, this is possible only for the smaller test cases, “Motiv.”, TC5 and TC6. For the larger test case TC7 and the realistic test case “Auto.”, we also have to optimize the routing of the TT flows in order to obtain schedulable solutions.

The conclusion is that only by using our *JRS* approach that takes the AVB into account during the routing and scheduling optimization, we are able to obtain schedulable solutions where all AVB flows are also schedulable. In addition, our approach is able to handle large problem sizes, such as TC7, with a network of 402 devices and 891 flows.

4.7 Conclusions

In this paper we have considered TSN-based cyber-physical systems and running mixed-criticality real-time applications that use both TT and AVB traffic. We were interested to decide the routing, the number of TT queues, the allocation of TT flows to TT queues and the GCLs of TT queues such that all flows are schedulable. Our approach was to use an integrated heuristic-metaheuristic, called Joint Routing and Scheduling (JRS), to search for TT routing and scheduling solutions where both TT and AVB flows are schedulable. Each solution visited during the search was evaluated in terms of AVB schedulability using a Network Calculus approach that takes into account the impact of TT GCLs on the WCDs of AVB flows. Our results show that only by taking into account the AVB flows during the routing and GCL synthesis we can obtain schedulable solutions for both TT and AVB.

Bibliography

[Aer09] Aeronautical Radio, Inc. ARINC 664P7: Aircraft Data Network, Part 7, Avionics Full-DupleX Switched Ethernet Network (AFDX), 2009.

[AGRN16] Guy Avni, Shibashis Guha, and Guillermo Rodriguez-Navas. Synthesizing Time-Triggered Schedules for Switched Networks with Faulty Links. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*, pages 1–10, 2016.

[ALAS15] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach.* MIT Press, 2015.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[AMF12] Hamdi Ayed, Ahlem Mifdaoui, and Christian Fraboul. Frame packing strategy within gateways for multi-cluster avionics embedded networks. In *Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8, 2012.

[APLB13] Giuliana Alderisi, Gaetano Patti, and Lucia Lo Bello. Introducing support for scheduled traffic over IEEE audio video bridging networks. In *Proceedings of IEEE Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–9, 2013.

[Apt03] Krzysztof Apt. *Principles of constraint programming.* Cambridge U. Press, 2003.

[ART14] Bjoern Annighoefer, Caroline Reif, and Frank Thieleck. Network topology optimization for distributed integrated modular avionics. In *Proceedings of IEEE/AIAA Digital Avionics Systems Conference (DASC)*, pages 4A1–1–4A1–12, 2014.

[ASBCH13] Ahmad Al Sheikh, Olivier Brun, Maxime Cheramy, and Pierre-Emmanuel Hladik. Optimal design of virtual links in AFDX networks. *Real-time Systems*, 49(3):308–336, 2013.

[AVB05] AVB Task Group. Audio-Video Bridging (AVB), 2005.

[AVB09] AVB Task Group. IEEE 802.1Qav/D7.0: Forwarding and Queuing Enhancements for Time-Sensitive Streams, 2009.

[AVB11] AVB Task Group. IEEE 802.1ba/D2.5: Audio Video Bridging (AVB) Systems, 2011.

[BK14] Edmund K Burke and Graham (Eds.) Kendall. *Search methodologies.* Springer, 2014.

[BLAC05] Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems.* Springer, 2005.

[But11] Giorgio Buttazzo. *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Springer Publishing Company, Incorporated, 3rd edition, 2011.

[Con06] FlexRay Consortium. Flexray, 2006.

[CRE$^+$12] Rodney Cummings, Kai Richter, Rolf Ernst, Jonas Diemer, and Arkadeb Ghosal. Exploring use of ethernet for in-vehicle control applications: AFDX, TTEthernet, EtherCAT, and AVB. *SAE Technical Papers*, 5(1):72–88, 2012.

[CS16] 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.

[CSCS16] Silviu S. Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling real-time communication in IEEE 802.1Qbv time sensitive networks. In *proceedings of International Conference on Real-Time Networks and Systems (RTNS)*, pages 183–192, 2016.

[DAB14] Joan Adrià Ruiz De Azua and Marc Boyer. Complete Modelling of AVB in Network Calculus Framework. In *Proceedings of International Conference on Real-Time Networks and Systems (RTNS)*, pages 55–64, 2014.

[Dec05] Jean-Dominique Decotignie. Ethernet-based real-time and industrial communications. *Proceedings of the IEEE*, 93(6):1102–1117, 2005.

[DN16] Frank Duerr and Naresh Ganesh Nayak. No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN). In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 203–212, 2016.

[Est07] Jeff A. Estefan. Survey of model-based systems engineering (MBSE) methodologies. *INCOSE MBSE Focus Group*, 25(8):1–12, 2007.

[Eth03] EtherCAT Technology Group. EtherCAT (Ethernet for Control Automation Technology), 2003.

[Eth18] EtherCAT Technology Group (ETG). EtherCAT Approach Supported by Key TSN Switch Vendors, 2018.

[Fel04] Joachim Feld. PROFINET—scalable factory communication for all applications. In *Proceedings of the IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 33–38, 2004.

[FMHH01] Thomas Fuehrer, Bernd Mueller, Florian Hartwich, and Robert Hugel. Time Triggered CAN (TTCAN). In *SAE World Congress*, page 7s. SAE International, 2001.

[FR89] Thomas A Feo and Mauricio G. C. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, 8(2):67–71, 1989.

[GHKS98] Miltos D. Grammatikakis, D. Frank Hsu, Miro Kraetzl, and Jop F. Sibeyn. Packet Routing in Fixed-Connection Networks: A Survey. *Journal of Parallel and Distributed Computing*, 54(2):77–132, 1998.

[GJS76] Michael R. Garey, David S. Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.

[GP16] Voica Gavrilut and Paul Pop. Traffic class assignment for mixed-criticality frames in TTEthernet. *ACM Sigbed Rev.*, 13(4):31–36, 2016.

[GP18] Voica Gavrilut and Paul Pop. Scheduling in time sensitive networks (TSN) for mixed-criticality industrial applications. In *Proceedings of International Workshop on Factory Communication Systems (WFCS)*, pages 1–4, 2018.

[GPew] Voica Gavrilut and Paul Pop. Traffic Type Assignment for TSN-based Mixed-Criticality Cyber-Physical Systems. *ACM Transactions on Cyber-Physical Systems (TCPS)*, page 20s, 2018 (in review).

[GTSP15] Voica Gavrilut, Domitian Tamas-Selicean, and Paul Pop. Fault-Tolerant Topology Selection for TTEthernet Networks. In *Proceedings of the Safety and Reliability of Complex Engineered Systems Conference*, pages 4001–4009, 2015.

[Gur] Gurobi Optimizer. Mathematical Programming Solver.

[GZPS17] Voica Gavrilut, Bahram Zarrin, Paul Pop, and Soheil Samii. Fault-Tolerant Topology and Routing Synthesis for IEEE Time-Sensitive Networking. In *Proceedings of International Conference on Real-Time Networks and Systems (RTNS)*, pages 267–276, 2017.

[GZRPss] Voica Gavrilut, Luxi Zhao, Michael L. Raagaard, and Paul Pop. AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN. *IEEE Access*, page 14s, 2018 (in press).

[Hap16] Julien Happich. Time sensitive networks approaches the car, 2016.

[HKGF09] Thomas Herpel, Bernhard Kloiber, Reinhard German, and Steffen Fey. Routing of Safety-Relevant Messages in Automotive ECU Networks. In *Proceedings of the Vehicular Technology Conference (VTC)*, pages 1–5, 2009.

[Hon92] Honeywell. SAFEbus, 1992.

[HSF14] Tasnim Hamza, Jean-Luc Scharbarg, and Christian Fraboul. Priority assignment on an avionics switched Ethernet Network (QoS AFDX). In *Proceedings of International Workshop on Factory Communication Systems (WFCS)*, pages 1–8, 2014.

[HZL17] Feng He, Lin Zhao, and Ershuai Li. Impact Analysis of Flow Shaping in Ethernet-AVB/TSN and AFDX from Network Calculus and Simulation Perspective. *Sensors*, 17(5), 2017.

[IEE10] IEEE. IEEE 802.1Qat - IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol, 2010.

[IEE12] IEEE. IEEE 802.3 - IEEE Standard for Ethernet, 2012.

[Int98] International Electrotechnical Commission (IEC). Fieldbus, 1998.

[Int01] Profibus & Profinet International. Profinet, 2001.

[JB04] Dirk Jansen and Holger Buttner. Real-time Ethernet: the EtherCAT solution. *Computing and Control Engineering*, 15(1):16–21, 2004.

[KG93] Hermann Kopetz and Guenther Gruensteidl. TTP - A time-triggered protocol for fault-tolerant real-time systems. In *Proceedings of International Symposium on Fault-Tolerant Computing (FTCS)*, pages 524–533, 1993.

[Kni02] Jonathan Knight. Safety critical systems: challenges and directions. In *Proceedings of the International Conference on Software Engineering (ICSE)*, pages 547–550, 2002.

[Kop91] Hermann Kopetz. Event-triggered versus time-triggered real-time systems. *Lecture Notes in Computer Science*, 563:87–101, 1991.

[Kop11] Hermann Kopetz. *Real-time systems : Design principles for distributed embedded applications*. Springer, 2011.

[KP00] Way Kuo and V Rajendra Prasad. An annotated overview of system-reliability optimization. *IEEE Transactions on Reliability*, 49(2):176–187, 2000.

[KRD02] Nicolas Krommenacker, Eric Rondeau, and Thierry Divoux. Genetic algorithms for industrial ethernet network design. In *Proceedings of IEEE International Workshop on Factory Communication Systems (WFCS)*, pages 149–156, 2002.

[KS06] Abdullah Konak and Alice E Smith. Network reliability optimization. In *Handbook of Optimization in Telecommunications*, pages 735–760. Springer, 2006.

[Lau16] Laursen, Sune Molgaard and Pop, Paul and Steiner, Wilfried. Routing Optimization of AVB Streams in TSN Networks. *ACM Sigbed Review*, 13:43–48, 2016.

[LCM11] Martin Lukasiewycz, Samarjit Chakraborty, and Paul Milbredt. FlexRay Switch Scheduling - A Networking Concept for Electric Vehicles. In *Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, 2011.

[LSS$^+$12] Martin Lukasiewycz, Sebastian Steinhorst, Florian Sagstetter, Wanli Chang, Peter Waszecki, Matthias Kauer, and Samarjit Chakraborty. Cyber-physical systems design for electric vehicles. In *Proceedings of Euromicro Conference on Digital System Design (DSD)*, pages 477–484, 2012.

[MAR08] Vaclav Mikolasek, Astrit Ademaj, and Stanislav Racek. Segmentation of standard ethernet messages in the time-triggered ethernet. In *Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 392–399, 2008.

[ML09] Muriel Medard and Steven S. Lumetta. Network reliability and fault tolerance. Technical report, Massachusetts Institute of Technology, 2009.

[MLC15] Renato Mancuso, Andrew V. Louis, and Marco Caccamo. Using traffic phase shifting to improve AFDX link utilization. In *Proceedings of International Conference on Embedded Software (EMSOFT)*, pages 256–265, 2015.

[MS17] Dorin Maxim and Ye-Qiong Song. Delay Analysis of AVB traffic in Time-Sensitive Networks (TSN). In *Proceedings of International Conference on Real-Time Networks and Systems (RTNS)*, pages 18–27, 2017.

[MVNB] Jorn Migge, Josetxo Villanueva, Nicolas Navet, and Marc Boyer. Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks.

[Nay17] Nayak, Naresh Ganesh and Duerr, Frank and Rothermel, Kurt. Routing Algorithms for IEEE802.1Qbv Networks. *ACM Sigbed Review*, x:6, 2017.

[NSS00] Nicolas Navet, Y-Q Song, and François Simonot. Worst-Case Deadline Failure Probability in Real-Time Applications Distributed over Controller Area Network. *Journal of Systems Architecture*, 46(7):607–617, 2000.

[On-14] On-road Automated Vehicle Standards Committee. SAE J3016: Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems. Technical report, SAE International, 2014.

[PA04] Paulo Pedreiras and Luis Almeida. Message routing in multi-segment FTT networks: the isochronous approach. In *Proceedings of Parallel and Distributed Processing Symposium (PDPS)*, pages 122–129, 2004.

[PD12] Quang Dung Pham and Yves Deville. Solving the quorumcast routing problem by constraint programming. *Constraints*, 17(4):409–431, 2012.

[PEP05] Paul Pop, Petru Eles, and Zebo Peng. Schedulability-driven frame packing for multicluster distributed embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 4(1):112–140, 2005.

[PGAB05] Paulo Pedreiras, Paolo Gai, Luis Almeida, and Giorgio 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.

[PGPE11] Paul Pop, Alois Goller, Traian Pop, and Petru Eles. Development tools. In Roman Obermaisser, editor, *Time-Triggered communication*, chapter 7, pages 363–522. CRC Press, 2011.

[PPEP08] 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.

[PRCS16] Paul Pop, Michael L. Raagaard, Silviu S. Craciunas, and Wilfried Steiner. Design Optimization of Cyber-Physical Distributed Systems using IEEE time-sensitive Networks (TSN). *IET Cyber-Physical Systems: Theory & Applications*, 1(1):86–94, 2016.

[PSRH15] Francisco Pozo, Wilfried Steiner, Guillermo Rodríguez-Navas, and Hans Hansson. A decomposition approach for SMT-based schedule synthesis for time-triggered networks. In *Proceedings of IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8, 2015.

[PSS15] Wolfgang Puffitsch, Rasmus Bo Sorensen, and Martin Schoeberl. Time-division multiplexing vs network calculus. In *proceedings of International Conference on Real-Time Networks and Systems (RTNS)*, pages 289–296, 2015.

[PTO18] Maryam Pahlevan, Nadra Tabassam, and Roman Obermaisser. Heuristic list scheduler for time triggered traffic in time sensitive networks. *ACM Sigbed Review*, x:1–6, 2018.

[Rob86] Robert Bosch GmbH. Controller area network (can), 1986.

[RP17] Michael Lander Raagaard and Paul Pop. Optimization algorithms for the scheduling of IEEE 802.1 Time-Sensitive Networking (TSN). Technical report, Technical University of Denmark, January 2017.

[RR14] Mauricio GC Resende and Celso C Ribeiro. GRASP: greedy randomized adaptive search procedures. In *Search methodologies*, pages 287–312. Springer, 2014.

[Rus01] John Rushby. A comparison of bus architectures for safety-critical embedded systems. Technical report, Computer Science Laboratory, SRI International, 2001.

[SAE11] SAE. AS6802: Time-Triggered Ethernet, 2011.

[SCS14] Ramon Serna Oliver, Silviu S Craciunas, and Georg Stöger. Analysis of deterministic ethernet scheduling for the industrial internet of things. In *Proceedings of the IEEE International Workshop on*

Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), pages 320–324, 2014.

[SCS18] Ramon Serna Oliver, Silviu S. Craciunas, and Wilfried Steiner. IEEE 802.1Qbv Gate Control List Synthesis using Array Theory Encoding. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018.

[SDT+17] Eike Schweissguth, Peter Danielis, Dirk Timmermann, Helge Parzyjegla, and Gero Mühl. ILP-based joint routing and scheduling for time-triggered networks. In *Proceedings of the International Conference on Real-Time Networks and Systems (RTNS)*, pages 8–17, 2017.

[SG12] Stefan Schneele and Fabien Geyer. Comparison of IEEE AVB and AFDX. In *Proceedings of the IEEE/AIAA Digital Avionics Systems Conference (DASC)*, pages 7A7–1–7A7–9, 2012.

[Sha98] Paul Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP)*, pages 417–431, 1998.

[Sie04] Siemens AG. SN 29500-5: Failure rates of components expected values. Section 5. Expected values for electrical connections, connectors and sockets. Technical Report SN 29500-5, Siemens AG, 2004.

[SN06] Rishi Saket and Nicolas Navet. Frame packing algorithms for automotive applications. *Journal of Embedded Computing*, 2(1):93–102, 2006.

[SS16] Johannes Specht and Soheil Samii. Urgency-Based Scheduler for Time-Sensitive Switched Ethernet Networks. In *Proceedings of Euromicro Conference on Real-Time Systems (ECRTS)*, pages 75–85, 2016.

[SS17] Johannes Specht and Soheil Samii. Synthesis of Queue and Priority Assignment for Asynchronous Traffic Shaping in Switched Ethernet. In *Proceedings of IEEE Real-Time Systems Symposium (rtss)*, pages 178–187, 2017.

[Ste10] Wilfried Steiner. An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In *Proceedings of Real-Time Systems Symposium (RTSS)*, pages 375–384, 2010.

[Ste11] Wilfried Steiner. Synthesis of static communication schedules for mixed-criticality systems. In *Proceedings of IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, pages 11–18, 2011.

[Ste16] Wilfried Steiner. Deterministic ethernet for real-time and critical applications. `"http://www.av.it.pt/wfcs2016/admin/files/Keynotes/Wilfried%20Steiner/2016-05-04-Deterministic-Ethernet.pdf"`, 2016. Keynotes of International Workshop on Factory Communication Systems (WFCS).

[TE16] Daniel Thiele and Rolf Ernst. Formal worst-case performance analysis of time-sensitive ethernet with frame preemption. In *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–9, 2016.

[Tib13] Gissila Tibebu. Connectors and vibrations – damages in different electrical environments. Master's thesis, Blekinge Institute of Technology, 2013.

[TSN12] TSN Task Group. Time-Sensitive Networking (TSN), 2012.

[TSN15] TSN Task Group. IEEE 802.1Qbv/D3.1: Enhancements for Scheduled Traffic, 2015.

[TSN17] TSN Task Group. IEEE 802.1CB/D2.8: Frame Replication and Elimination for Reliability, 2017.

[TSN18a] TSN Task Group. IEEE p802.1Qcc/D2.3: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements, 2018.

[TSN18b] TSN Task Group. IEEE p802.1Qcr/D0.4: Asynchronous Traffic Shaping, 2018.

[TSPM15] Domitian Tamas-Selicean, Paul Pop, and Jan Madsen. *Design of Mixed-Criticality Applications on Distributed Real-Time Systems*. PhD thesis, 2015.

[TSPS14] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. Design optimization of TTEthernet-based distributed real-time systems. *Real-Time Systems*, 51(1):1–35, 2014.

[TSPS15] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. Design optimization of TTEthernet-based distributed real-time systems. *Real-Time Systems*, 51(1):1–35, 2015.

[TW11] Andrew S. Tanenbaum and David J. Wetherall. *Computer networks*. Prentice-Hall,, 2011.

[Ull75] Jeffrey D. Ullman. NP-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[Uni88] United States Department of Transportation (DOT). Advisory Circular (AC) 25.1309-1A - System Design and Analysis, 1988.

[Val79] Leslie G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[Veh93] Vehicle Architecture For Data Communications Standards Committee. SAE J2056/1: Class C Application Requirement Considerations. Technical report, SAE International, 1993.

[vOPF16] Nikolaj van Omme, Laurent Perron, and Vincent Furnon. OR-Tools User's Manual. Technical report, Google, 2016.

[WH00] Bin Wang and Jennifer C. Hou. Multicast Routing and its QoS Extension: Problems, Algorithms, and Protocols. *IEEE Network*, 14(1):22–36, 2000.

[Yen71] Jin Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[ZP18] Cristian Zara and Paul Pop. Algorithms for the optimization of safety-critical networks. Master's thesis, Technical University of Denmark, 2018.

[ZPL$^+$17] Luxi Zhao, Paul Pop, Qiao Li, Junyan Chen, and Huagang Xiong. Timing analysis of rate-constrained traffic in ttethernet using network calculus. *Real-time Systems*, 53(2):254–287, 2017.

[ZPZL18] Luxi Zhao, Paul Pop, Zhong Zheng, and Qiao Li. Timing analysis of AVB traffic in TSN networks using network calculus. In *Proceedings of the Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 25–36, 2018.