Scheduling and Optimization for Resource Management in Novel Applications in Communication and Energy Systems

Dissertation
der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Thomas Stüber
aus Tübingen

Tübingen
2024

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	4. Oktober 2024
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Michael Menth
2. Berichterstatter:	Prof. Dr. Klaus-Jörn Lange

Contents

List of Abbreviations

AVB	Audio Video Bridging
BE	best-effort
BFIR	Bit Forwarding Ingress Router
BFER	Bit Forwarding Egress Router
BIER	Bit Index Explicit Replication
CCDF	complementary cumulative distribution function
DHW	Domestic Hot Water
FIFO	first-in-first-out
GCL	Gate Control List
IETF	Internet Engineering Task Force
IPMC	IP multicast
MILP	Mixed Integer Linear Program
PSFP	Per-Stream Filtering and Policing
PV	Photovoltaic
QoS	Quality of Service
SD	Subdomain
SDI	Subdomain Identifier
SDN	Software-Defined Networking
SEET	Segment-Encoded Explicit Tree
SMT	Satisfiability Modulo Theories
TAS	Time-Aware Shaper
TSA	Transmission Selection Algorithm
TSN	Time-Sensitive Networking
VLAN	Virtual Local Area Network

Danksagung

Das Verfassen der vorliegenden wissenschaftlichen Arbeit wäre ohne die Hilfe und Unterstützung vieler Menschen nicht möglich gewesen. Besonders hervorheben möchte ich an dieser Stelle meine Eltern, die mich seit meiner Kindheit stets unterstützt und mein Interesse gefördert haben. Des Weiteren haben viele Freunde und mein Bruder meine Art und meine Angewohnheiten nicht nur hingenommen, sondern mir auch bei diversen großen und kleinen Problemen geholfen. Zu guter Letzt möchte ich mich bei meiner Frau, Stefanie, bedanken, die mir erst die Motivation gab das Promotionsvorhaben erfolgreich zu Ende zu bringen. Diese Arbeit ist ihr gewidmet, weil wir auch in schweren Zeiten immer zusammenhalten.

Summary

Abstract

Resource management is a generic term for the process of determining the most beneficial way to employ some limited set of resources. Typically, not all demands for a set of resources can be fulfilled and determining their most efficient usage is challenging. To make matters worse, many problems are not static. Instead, the state of a system changes dynamically and depends on the usage of its resources in time. Planning the usage of resources in time is denoted as scheduling in the literature. Computing schedules is hard and requires the accurate modelling of the underlying system.

The objective of this work is the development and evaluation of scheduling algorithms for novel applications in power systems, real-time communication, and multicast protocols. We contribute various research works about scheduling in power systems. We propose a scheduling algorithm to save energy costs in production processes, evaluate forecasting methods for domestic demands, and present an optimization algorithm to maximize self-consumption of roof-top PV energy with a heat pump. In the domain of real-time communication, we review the state of the art of scheduling algorithms for the so-called Time-Aware Shaper (TAS). We report results of a quantitative study comparing various scheduling algorithms for TAS. Additionally, we propose a scheduling algorithm that computes schedules robust against some sources of non-determinism. In the field of multicast protocols, we first contribute a partitioning algorithm needed to scale Bit Indexed Explicit Replication (BIER) to large networks. Then, we present Segment-Encoded Explicit Trees (SEET), a novel multicast protocol that allows tree engineering.

The research presented in this thesis has been funded by different research projects by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2, the German Federal Ministry of Education and Research (BMBF) under support code 16KIS1161 (Collaborative Project KITOS), and the German Federal Ministry for Economic Affairs and Energy 16KN039521 (ZIM).

Kurzfassung

Ressourcen-Management ist der Prozess zur Bestimmung der besten Verwendung von begrenzten Ressourcen. Typischerweise kann nicht der gesamte Bedarf an Ressourcen gedeckt werden und das Bestimmen der effizientesten Verwendung ist herausfordernd. Darüber hinaus sind viele Probleme in diesem Bereich dynamisch. Das bedeutet, dass sich die Verwendung der Resourcen und der Zustand des Systems in der Zeit ändern kann. Das zeitliche Planen der Verwendung von Ressourcen wird in der Literatur als Scheduling bezeichnet. Das Berechnen von Schedules (engl. *Ablaufpläne*) ist oft schwierig und erfordert eine korrekte Modellierung des zugrundeliegenden Systems.

Das Ziel dieser Arbeit ist die Entwicklung und Evaluation von Planungsalgorithmen für neuartige Anwendungen in den Bereichen Echtzeitkommunikation, Multicast Protokolle, und Energiesysteme. Im Bereich der Energiesysteme stellen wir einen Planungsalgorithmus zur Einsparung von Energiekosten in Produktionsprozessen vor, werten Vorhersagealgorithmen für Energiebedarfe in Wohnhäusern aus, und präsentieren ein Optimierungsverfahren mit dem der Eigenverbrauch von PV Energie durch Wärmepumpen optimiert werden kann. Im Bereich Echtzeitkommunikation wurde der aktuelle Forschungsstand zu Planungsalgorithmen für den sogenannten Time-Aware Shaper (TAS) zusammengefasst. Außerdem stellen wir die Ergebnisse einer quantitativen Vergleichsstudie zu mehreren Planungsalgorithmen für TAS vor. Darüber hinaus präsentieren wir einen Planungsalgorithmus, der robuste Ablaufpläne für den Fall von nicht-deterministischem Verhalten berechnen kann. Im Bereich von Multicast Protokollen stellen wir zuerst einen Algorithmus vor, der bnötigt wird, um Bit Indexed Explicit Replication (BIER) für große Netzwerke zu skalieren. Außerdem führen wir Segment-Encoded Explicit Trees (SEET) ein. Dabei handelt es sich um ein neuartiges Protokoll für zustandslose Multicast Kommunikation, das Tree Engineering unterstützt.

Die in dieser Arbeit vorgestellten Forschungsergebnisse wurden im Rahmen verschiedener Forschungsprojekte von der Deutschen Forschungsgemeinschaft (DFG) unter den Förderkennzeichen ME2727/1-2, dem Bundesministerium für Bildung und Forschung (BMBF) unter dem Förderkennzeichen 16KIS1161 (Verbundprojekt KITOS), und dem Bundesministerium für Wirtschaft und Energie unter Förderkennzeichen 16KN039521 (ZIM) gefördert.

List of Publications

The individual contributions to all publications (§ 6 Abs. 2 Satz 3 der Promotionsordnung) can be found in the appendix.

Accepted Manuscripts (Core Content)

1. Thomas Stüber, Florian Heimgärtner, and Michael Menth. **Day-Ahead Optimization of Production Schedules for Saving Electrical Energy Costs** [SHM19]. *Proceedings of the Tenth ACM International Conference on Future Energy Systems (e-Energy '19)*, Phoenix, USA, pp. 192–203, 2019. The author version of this publication can be found in the Appendix 1.1. The paper is also available online at the following URL: `https://doi.org/10.1145/3307772.3328302`

2. Thomas Stüber, Lukas Osswald, Steffen Lindner, and Michael Menth. **A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN)** [SOLM23]. *IEEE Access*, vol. 11, pp. 61192-61233, 2023. The author version of this publication can be found in the Appendix 1.2. The paper is also available online at the following URL: `https://doi.org/10.1109/ACCESS.2023.3286370`

3. Thomas Stüber, Manuel Eppler, Lukas Osswald, and Michael Menth. **Performance Comparison of Offline Scheduling Algorithms for the Time-Aware Shaper (TAS)** [SEOM24]. *IEEE Transaction on Industrial Informatics*, early access. The most recent version of this publication can be found in the Appendix 1.3.

4. Daniel Merling, Thomas Stüber, and Michael Menth. **Efficiency of BIER Multicast in Large Networks** [MSM23]. *IEEE Transactions on Network and Service Management (TNSM)*, vol. N/A, pp. N/A, 2023. The author version of this publication can be found in the Appendix 1.4. The paper is also available online at the following URL: `https://doi.org/10.1109/TNSM.2023.3262294`

Submitted Manuscripts (Core Content)

5. Thomas Stüber, Bernd Thomas, and Michael Menth. **Minimizing Grid Electricity Consumption and On-/Off-Cyles for Heat Pumps in Single-Family Homes with PV Panels** [STM23]. Submission to the *Applied Energies* journal on 2023-09-27. Under Review. The most recent version of this publication can be found in the Appendix 2.1.

6. Thomas Stüber, Lukas Osswald, and Michael Menth. **Efficient Robust Schedules (ERS) for Time-Sensitive Networking** [SOM24]. Submission to the *IEEE Transactions on Network and Service Management* journal on 2023-10-23. Under Review. The most recent version of this publication can be found in the Appendix 2.2.

7. Steffen Lindner, Thomas Stüber, Toerless Eckert, and Michael Menth. Scalability of Segment-Encoded Explicit Trees (SEETs) for Efficient Stateless Multicast. The most recent version of this publication can be found in the Appendix 2.3.

Accepted Manuscripts (Additional Content)

8. Thomas Stüber, Ricarda Hogl, Bernd Thomas, and Michael Menth. **Comparison of Forecasting Methods for Energy Demands in Single Family Homes** [SHTM21]. *ETG Congress 2021*, Wuppertal, Germany, pp. 1–5, 2021. The author version of this publication can be found in the Appendix 3.1.

9. Florian Heimgärtner, Sascha Heider, Thomas Stüber, Daniel Merling, and Michael Menth. **Load Profile Negotiation for Compliance with Power Limits in Day-Ahead Planning** [HHS+19]. *International ETG-Congress 2019*, Esslingen, Germany, pp. 1–6, 2021. The author version of this publication can be found in the Appendix 3.2.

1 Introduction & Overview

Resource management is a generic term for the process of determining how some limited set of resources can be employed efficiently and with the most benefit. Typical resources in communication networks are transmission bandwidth and exclusive access to a medium. Energy systems feature different resources such as roof-top PV power and heat storage. However, despite the different nature of the featured resources, similar methods can be applied to problems from both domains. This thesis focuses mainly on scheduling problems, i.e., problems about planning the usage of resources in time.

In the following, we give an overview of the research presented in this thesis. First, we introduce the research objectives. Then, we describe the research context. Finally, we discuss the major results and findings.

1.1 Research Objective

Distributed systems combine resources of logically separate units, e.g., network devices or production machines. The usage of these resources must be coordinated to gain some benefit or to operate the distributed system at all. This coordination can be done decentrally, i.e., every unit decides for itself how it should operate, or centrally, i.e., a central process plans the usage of resources. This thesis focuses on the latter case, i.e., planning problems in distributed systems with central control. These problems are tackled with established formal methods from the scheduling and optimization domains. We remark that this thesis does not propose new formal methods, but that existing methods are applied to problems from novel application. These methods are employed to conduct quantitative evaluations for complex distributed systems. The research objective of this thesis is to improve the state of the art in planning algorithms for two kinds of distributed systems and to gain understanding about these systems by rigorous evaluation.

The first objective is the development and evaluation of scheduling algorithms for novel applications in power systems. Volatile energy prices, renewable power production, and techniques denoted as demand side management result in incentives to shift energy-intensive tasks in time. The presented scheduling algorithms aim to reduce energy costs or consumption by leveraging flexibility on when these tasks are executed.

The second objective is the development of scheduling and optimization algorithms for novel applications in communication networks. This objective is divided into two categories. First, we investigate scheduling algorithms for Time-Sensitive Networking [80216]. Time-Sensitive Networking is an enhancement for bridged Ethernet networks. It enables the scheduling of frame transmissions in time to give real-time guarantees for individual streams and frames. Second, we evaluate novel protocols for stateless multicast transmissions. To that end, we develop optimization algorithms to use these protocols efficiently. We investigate Bit Indexed Explicit Replication (BIER), a stateless multicast protocol proposed by the IETF [WRD+17], and Segment Encoded Explicit Trees (SEET), a novel approach proposed in this thesis.

1.2 Research Context

The research presented in this thesis has been funded by different research projects by the German Federal Ministry of Education and Research (BMBF) under support codes 16KIS1161 (Collaborative Project KITOS), and the German Federal Ministry for Economic Affairs under support code 16KN039521 (ZIM). Further, some work has been funded by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The published versions of the publications in the appendix indicate which work has been funded by which research project.

All research was carried out in collaboration with colleagues. A description of the contributions that my coworkers and I made to the individual works can be found in the appendix.

1.3 Research Results

This thesis comprises 6 publications and three works that are currently under review. All publications can be found in the appendix. Chapter 2 summarizes and presents the

research results of these publications. For each publication, the problem description and findings are presented. In the following, an overview of the research is given.

1.3.1 Energy Systems

The increasing deployment of renewable energy sources results in fluctuations in the production of electrical energy. Combined with time-dependent demand profiles for electrical energy, this leads to volatile energy prices over the course of a day due to demand and supply. Large consumers can benefit from these price fluctuations by participating in power exchanges, i.e., they order energy for times when it is cheap. We remark that times with low energy prices correspond to overproduction of energy due to renewable energy sources. The overproduction must be consumed to prevent harm to the power grid. If energy-intensive tasks can be shifted to these times, energy costs, i.e., money, can be reduced and the adverse effects of overproduction are mitigated. However, most energy-intensive tasks do not run in isolation but depend on other tasks in larger production processes.

The planning of all tasks in such processes is a complex scheduling problem. In Stüber et al. [SHM19], we show that deciding whether a schedule for a production process exists is an NP-complete problem. However, we are not only interested in finding some schedule for production processes, but we also want to find a schedule that minimizes energy costs. Thus, we apply Mixed Integer Linear Programming (MILP), an established method from mathematical optimization. We reduce the problem of computing an optimal production schedule to MILP solving, i.e., for a given production process we construct a MILP that captures the solution space of the scheduling problem instance. The work of Stüber et al. [SHM19] (cf. Appendix 1.1) presents this construction. Additionally, we employ the constructed MILPs in a case study to evaluate the possible energy cost savings of a cement plant. This study reveals that about 8-12% of the energy costs can be saved by shifting energy-intensive tasks in time. Thus, the study demonstrates that formal methods can contribute to the energy transition. This work is discussed in Section 2.1.1.

The scheduling model of Stüber et al. [SHM19] is rather general and can be applied to other problems. We employ the construction with slight modifications to schedule the operation of heat pumps in domestic buildings in [STM23] (cf. Appendix 2.1). German politics promoted the deployment of heat pumps and PV panels in single family homes during the last decade. By self-consumption of roof-top PV energy, heat pumps can

be operated more efficiently. We employ the MILP model to compute schedules that minimize additional energy purchase from the grid. However, lifetime and maintenance costs of heat pumps depend on the number of on/off cycles per year. Thus, we consider a multi-criterion optimization problem with short-term goal (minimize grid energy) and long-term goal (minimize on/off cycles). The two optimization goals are contradictory. We show that both goals can be balanced even though only short-term scheduling is possible during operation. Additionally, we quantify the additional energy costs and on/off cycles imposed by using forecasts instead of ground truth. We demonstrate that even a naive forecasting strategy can give good results with rather small penalties. Thus, formal tools can be applied successfully for the heat transition. This work is discussed in Section 2.1.2.

1.3.2 Communication Networks

At first glance, scheduling problems in communication networks seem to have few in common with energy systems. They utilize other resources and feature different constraints. However, scheduling is the process of determining when the available resource should be employed. The exact nature of resources does not matter from an abstract point of view in many cases. Thus, similar methods can be applied for scheduling problems in energy systems and communication networks.

1.3.2.1 Time-Sensitive Networking

Time-Sensitive Networking (TSN) is a set of standards that enhance bridged Ethernet networks for reliable communication with real-time requirements. Examples for real-time requirements are bounded latency and jitter for individual streams. TSN offers features for traffic scheduling. That means that the transmission of individual frames can be planned at sending and forwarding nodes such that real-time requirements are met. An introduction about TSN can be found in Section 2.2.1. The standards only describe the mechanisms needed for traffic scheduling but omits to define a scheduling algorithm. However, there is a large body of research that proposes more than 100 algorithms for this purpose.

The first contribution of this thesis with respect to TSN is a rigorous literature study about these algorithms. We discuss the main ideas and findings of each research work.

To that end, we classify research works by the respective major topic and the considered problem variation. For instance, works that focus on scheduling in presence of other traffic classes are discussed together. We give a tutorial about TSN and common algorithmic methodologies used in the literature to make the survey self-contained. Additionally, we compile important information about the results of all works in tabular form, e.g., network topologies, numbers of streams, and objective functions. Finally, we make recommendations for better scientific practices and highlight open problems. This survey is discussed in Section 2.2.2.

Based on the literature study we identified the lack of a quantitative comparison study as an important open problem. Many of the algorithms solve the same problem variation and only differ in the employed methodology. However, most researchers compare their algorithms only to outdated algorithms of themselves. Additionally, problem instances are neither described sufficiently nor published. To overcome these problems, we implement 11 well-known algorithms from the literature and perform extensive performance evaluations. To that end, we develop a set of problem instances for various parameter studies. The evaluations show that no algorithm is clearly superior over all other algorithms with respect to the metrics considered. However, we state recommendations about which algorithm to implement with respect to efficiency and implementation cost under different conditions. This work is discussed in Section 2.2.3.

Finally, we propose a scheduling algorithm based on MILPs that solves the scheduling problem under realistic conditions. In the context of the KITOS project, we found that most scheduling algorithms do not produce valid schedules for real hardware bridges, i.e., real-time requirements are violated. This is due to undesired non-deterministic effects that cannot be eliminated in practice. For instance, the clocks of different devices cannot be perfectly synchronized, and processing delays are neither constant nor deterministic. Additionally, the number of available GCL entries required to implement a schedule is limited in bridges. We propose an algorithm that computes robust schedules that can be implemented with a given number of GCL entries, i.e., all real-time requirements are guaranteed to be met even in case of non-deterministic effects. We demonstrate that a common countermeasure from literature, i.e., the introduction of large gaps between frames, imposes significant waste of bandwidth compared to the presented approach. This work is discussed in Section 2.2.4.

1.3.2.2 Multicast Protocols

Multicast denotes the concept of sending a message from one sender to multiple receivers. IPMC is the dominating multicast protocol on the Internet. It reduces packet transmissions compared to IP unicast by transmitting at most one packet copy per link. However, it requires the signaling of forwarding trees per multicast group and forwarding nodes must hold state. Thus, IPMC imposes significant overhead. The IETF proposed the stateless multicast protocol Bit Indexed Explicit Replication (BIER) for this reasons. BIER is a novel protocol that encodes receivers of a packet in a bitstring in the packet's header. The routing underlay of IP is employed to forward packets based on the bitstring. Thus, no signaling is required for forwarding trees of individual multicast groups. An introduction about BIER can be found in Section 2.2.5. Due to technical restrictions, the length of a packet header and thus the bitstring is limited in forwarding hardware. Therefore, the set of receivers in a network (BIER domain) must be divided into subsets (BIER subdomains) such that the header size is sufficient to represent all nodes in a subdomain. If a packet should be sent to receivers in different subdomains, one packet per domain must be sent. Thus, the partitioning of a network into subdomains affects various performance metrics such as the number of generated packets and the traffic transmitted in the network. We propose an algorithm based on greedy hill climbing that optimizes the partitioning of a network into subdomains. Additionally, we show that the heuristic computes near-optimal results in small networks. We conclude that we can employ the heuristic to evaluate the BIER mechanism itself. We show that while BIER imposes additional traffic compared to IPMC, it reduces traffic compared to IP unicast significantly. This work is discussed in Section 2.2.6.

Second, we present Segment Encoded Explicit Trees (SEET), a novel multicast protocol that offers similar advantages over IPMC than BIER but which also features tree engineering. SEET encodes the forwarding tree of a packet in the packet's header in so-called recursive units. We give an algorithm to construct the headers for SEET packets. Additionally, we present a quantitative comparison of SEET and BIER. We show that SEET outperforms BIER under most conditions even though BIER does not feature tree engineering. This work is discussed in Section 2.2.7.

2 Results & Discussion

This chapter summarizes and discusses the results of this thesis. Research results about algorithms for resource management in energy systems are elaborated in Section 2.1. We present a scheduling algorithm for production processes, evaluate forecasting models, and optimize self-consumption of PV energy with heat pumps. Afterwards, Section 2.2 presents research on algorithms for the management of resources in real-time networks and stateless multicast protocols. Research works in the domain of TSN are discussed in Section 2.2.1. We present an extensive survey about scheduling algorithms for the TAS, conduct a quantitative comparison study of several seminal works, and propose a novel scheduling algorithm to compensate for non-deterministic behavior. Section 2.2.5 highlights research results for multicast protocols. We propose a partitioning algorithm to scale BIER for large networks and a novel encoding to encode multicast trees in packet headers.

For each presented work, it is indicated whether they are part of the core content of this thesis or part of the additional content.

2.1 Algorithms for Resource Management in Energy Systems

The liberation of the energy markets and the transition to renewable energy sources result in the emergence of so-called smart grids. Smart grids feature various novel developments such as new pricing models, small-scale power generation, and demand side control. New pricing models allow to incentivize power consumption at times with overproduction or low consumption. Small-scale power generation can be employed to reduce energy costs on the consumer side. Demand side control is a consequence of the former two techniques and denotes the idea of controlling energy consumption to gain some benefit for all relevant parties, e.g., saving energy costs for consumers and maintaining a stable grid for grid operators. The following sections summarize research works that use one or more of these techniques.

2.1.1 Day-Ahead Optimization of Production Schedules for Saving Electrical Energy Costs

The section summarizes the contributions of the research work from Stüber et al. [SHM19]. This publication is part of the core content of this thesis. First, the research objective is introduced. Then, the scenario, the data set, and the algorithmic approach are presented. Finally, the results of the case study are summarized.

This work has been published as a conference paper at the ACM e-Energy conference [SHM19] (cf. Appendix 1.1) in 2019. It was presented in presence at the conference in Phoenix, AZ. Preliminary research for this publication, i.e., the development of the algorithm, was part of my Master's thesis.

2.1.1.1 Problem Description

Large volumes of energy are traded at power exchanges. These exchanges offer various pricing and delivery models for different time scales. The intra-day market offers the possibility to buy and sell electrical energy one day before delivery. The price per MWh depends on demand and supply. Working hours on the demand side and weather-dependent renewable energy sources on the production side result in volatile prices over the course of a day. If energy-intensive tasks can be shifted in time, this flexibility can be utilized to save energy costs. However, most energy intensive tasks do not run in isolation but depend on other tasks.

A production process consists of one or more devices that operate sequentially or in parallel. These machines consume and/or produce goods and electrical energy. The production and consumption rates are either fixed or linear to a modulation coefficient. Storages for different goods may be part of a production process and are the source or destination of production devices. They can be leveraged to decouple the operation of different devices from each other. Devices and storages are subject to various constraints such as maximum runtimes for devices or maximum filling states for storages. These constraints must hold at any time. A schedule of a production process is a description of the operation of all production devices during a time interval denoted as planning horizon. Typically, production goals state an amount of each good that should be produced during a planning horizon. A schedule is considered valid if all production goals do hold after the execution of the schedule.

The goal of this work is the development and evaluation of a scheduling algorithm for production processes. Given energy price data for the upcoming day, the algorithm computes a production schedule that minimizes energy costs such that all production goals are met. We use the algorithm to conduct a case study with the real production process of a cement plant.

2.1.1.2 Concept

We describe the algorithmic approach and the model and data set used in the case study.

2.1.1.2.1 Scheduling Algorithm Given a production process, including descriptions of all devices, storages, and their interconnections, and an energy price forecast, we construct an ILP model to compute optimized production schedules for the given process. The main idea of the model is to split the planning horizon into discrete units denoted as time slots. For each time slot, a production device may either run during the entire time slot, or it does not run within the time slot at all. Therefore, the production and consumption rates of devices are constant during a time slot, so storages are filled and depleted at linear rates. Thus, the transition between the system state before and after a time slot can be expressed by linear equations. The details of the model such as the variables and a formal description of the constraints can be found in Appendix 1.1.

We integrate the ILP model in a rolling horizon approach. This means that we compute production schedules for a planning horizon of multiple consecutive days, but use only a small fraction, e.g., a single day, of the schedules. After executing the schedule for this so-called control horizon, the planning horizon is shifted to start at the end of the control horizon, and a new schedule is computed. In this way, loads can be shifted between days and there is no incentive for the optimization to deplete storages at the end of the control horizon.

2.1.1.2.2 Model and Data Set We conduct a case study about the savings potential of a real cement plant with the presented algorithm. The specification of the production process, the properties of the involved devices, and the storage sizes were obtained from a joint research project with AVAT from Tübingen. Figure 2.1 depicts the production process. We use real energy price data from the year 2018 of Denmark obtained from

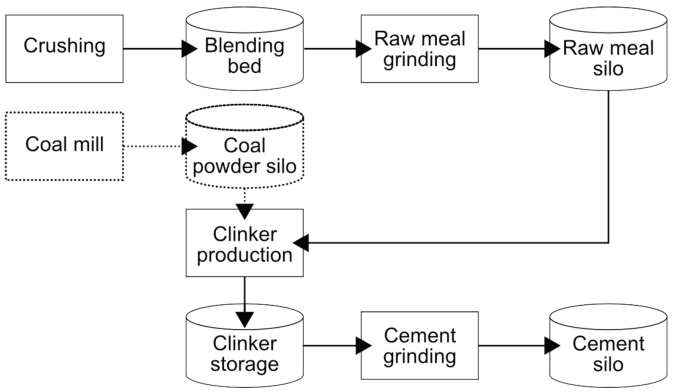


Figure 2.1: Model of the cement production process. Figure from [SHM19].

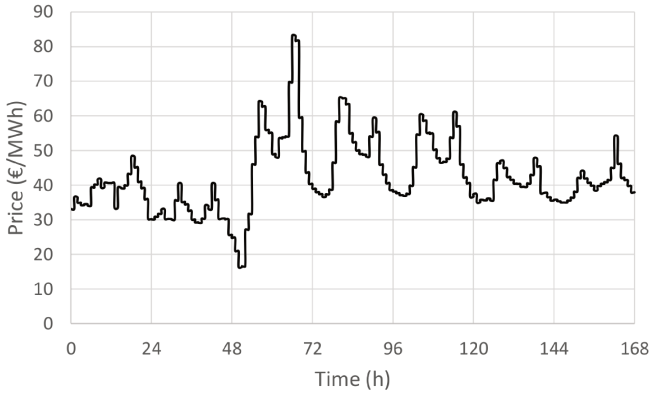


Figure 2.2: Example energy prices for the week from 2018-03-03 until 2018-03-09. Figure from [SHM19].

Nord Pool, a power exchange for central and northern Europe. Figure 2.2 shows the energy prices of the week from 2018-03-03 until 2018-09-09 as an example.

2.1.1.3 Case Study

We systematically evaluated the potential to save energy costs for the cement plant with different planning horizons. To that end, we defined a default schedule that corresponds to the schedule a human production engineer would design. This default schedule does not take energy price volatility into account, i.e., it is the same for every day of the year. We evaluated the rolling horizon approach by optimizing the entire year 2018 with zero, one, two, or six days look-ahead, and one day control horizon. We used absolute and relative energy cost savings compared to the default schedule to quantify the benefit of the proposed method. Table 2.1 compiles the total energy costs and savings for these different rolling horizon runs. Even without look-ahead, we observed a savings

Schedule	Total cost	Abs. sav. (€)	Rel. sav. (%)
Default	2,524,375.50	-	-
Opt. w/o look-ahead	2,323,047.60	201,327.89	7.98
Opt. w/ 1 day look-ahead	2,258,921.07	265,454.43	10.52
Opt. w/ 2 day look-ahead	2,242,216.13	282,159.38	11.18
Opt. w/ 6 day look-ahead	2,224,976.50	299,399.00	11.86

Table 2.1: Energy costs and savings with the proposed method compared to the default schedule in the year 2018. Adapted from [SHM19].

potential for about 200,000 € or 8% of the total energy costs per year in this specific scenario. With a small look-ahead of only one day, the saving compared to the default schedule can be increased to 10.5%. More look-ahead does not lead to significantly more cost savings compared to 1-day look-ahead. Additionally, forecasts for one day in advance are realistic for use cases of the proposed model. Thus, we recommend one day look-ahead.

Based on the rolling horizon method, we analyzed the impact of the scheduling flexibility to the savings potential. First, we showed that smaller storages lead to less energy savings. We introduced constraints that restrict the sizes of all storages to the maximum filling state of the respective storage in the default schedule. While one day look-ahead with the real storage sizes resulted in 10.52% cost savings compared to the default schedule, the same methodology lead to only 7.56% cost savings with reduced storages. This is due to less possibilities to shift energy intensive processes to earlier times with lower energy prices as not enough intermediate goods for later production stages can be produced in advance. Second, we showed that a high variability in energy prices results in a significantly higher savings potential compared to energy prices with low variability. To that end, we computed optimized schedules for the weeks with the highest and lowest standard deviation of energy prices in the year 2018. While low price variability still yields about 4% cost savings compared to the default schedule, high price variability leads to almost 20% cost savings.

2.1.1.4 Conclusion & Outlook

In this work, we proposed a scheduling algorithm to compute optimized operation schedules for energy-intensive production processes. We conducted an extensive case study for a cement plant. The case study revealed that significant cost savings in the range of 8%–12% compared to classical schedules are possible. However, we also

showed that the savings potential depends heavily on the potential to shift energy intensive processes to times with lower energy prices. We conclude that day-ahead optimization is a useful approach if the energy market offers price spreads and production processes have flexibility in time.

A drawback of the presented work is the use of ground truth price data as forecasts. It is possible that large quantitative forecasting errors still result in schedules that save a reasonable amount of energy costs when executed. Further studies may investigate the impact of these errors on energy savings.

The presented algorithm is not limited to production processes. For instance, it can be leveraged to compute heat pump schedules in domestic buildings with slight modifications. We present a research work about this topic based on the proposed scheduling algorithm in Section 2.1.2. Additionally, we quantify the impact of forecasting errors mentioned above in this work.

2.1.2 Minimizing Grid Electricity Consumption and On-/Off-Cyles for Heat Pumps in Single-Family Homes with PV Panels

The section summarizes the contributions of the research work from Stüber et al. [STM23]. This publication is part of the core content of this thesis. First, the research objective is introduced. Then, the scenario, the data set, and the algorithmic approach are presented. Finally, the results of the case study are summarized.

2.1.2.1 Problem Description

The German government incentivizes the installation of heat pumps in domestic homes due to climate change and global political conflicts. Additionally, the deployment of roof-top PV cells saw a rise over the last years for the same reasons. Self-consumption of the generated PV power can be employed to reduce the operation costs of a heat pump. However, PV energy is not sufficient to fulfill all demands in domestic homes such as consumer electronics, heat production, and domestic hot water (DHW) production. Moreover, PV production depends on weather, daytime, and season, and may vary considerably over the course of a day. Thus, computing operation schedules for the heat pump that minimize the additionally purchased grid energy is a challenge. To make matters worse, frequent on/off cycles reduce the lifespan of a heat pump. Minimizing the purchased grid energy increases the number of on/off cycles. This is due to

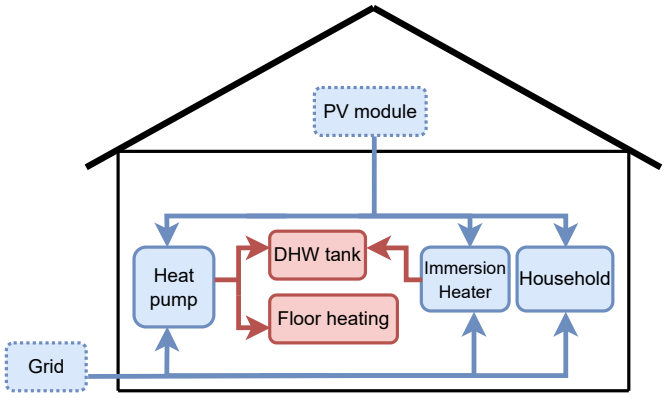


Figure 2.3: Depiction of the model used in the case study. Figure from [STM23].

the heat pump being turned off for short time intervals when there are no demands to fulfill. Instead, the number of on/off cycles can be reduced by operating the heat pump for long time intervals. However, the heat pump may run during suboptimal times in this case which increases the purchased grid energy. Thus, both objectives should be balanced by a control strategy. Additionally, a control strategy for heat pumps can only balance these objectives for a given planning horizon, i.e., a few days. Optimal control strategies for individual planning horizons may not balance the objectives when they are concatenated to form schedules for long time intervals, i.e., multiple months or years.

Figure 2.3 depicts a typical model of a single family home equipped with a heat pump and roof-top PV cells. At each time instant, the heat pump is either turned off, or operates in exactly one of two modes. The modes correspond to the production of heat and DHW. Additionally, the heat pump can be modulated between 30% and 100% of its maximum electrical and thermal power. Heat and DHW are stored in the building mass and a DHW tank. All heat and DHW demands of the residents must be satisfied from these storages. The heat pump can be operated with roof-top PV energy or with purchased energy from the grid. However, PV generation and residential demands are unknown in advance and must be forecast.

This work presents an algorithm to compute heat pump schedules, i.e., descriptions at which times the heat pump should run and how it should be modulated at these times. The algorithm computes schedules that minimize the additional purchased grid energy. We employ the algorithm to evaluate the benefit of optimized schedules in an extensive case study with a data set recorded from a real single family home. Additionally, we investigate the impact of forecasts on cost reductions.

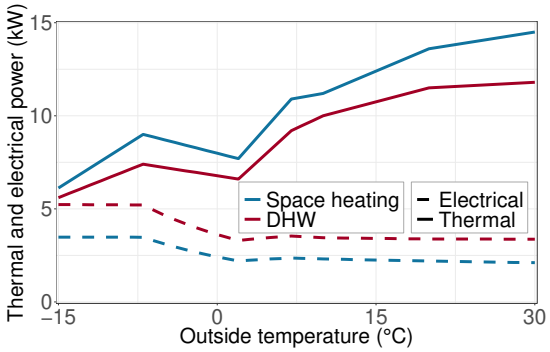


Figure 2.4: Electrical and thermal power of the heat pump with respect to outside temperature. Figure from [STM23].

2.1.2.2 Concept

We modified the ILP model from [SHM19] for this specific use case. This model was used to conduct an extensive case study for a real scenario which will be detailed in the following.

2.1.2.2.1 Model and Data Set We consider a residential building equipped with a heat pump and roof-top PV cells. Figure 2.3 depicts the components of the scenario used for the case study. The electrical and thermal power of the heat pump depend on the outside temperature. Figure 2.4 shows these relationships. The fraction of thermal and electrical power is denoted as Coefficient of Performance (COP). Typically, the COP is between 2 and 4 for common outside temperatures in Germany.

Heat and DHW are stored in the building mass and the DHW tank. We modelled thermal losses as exponential decay dependent on the state of charge of the respective storage. This means that the rate at which a storage is drained is the fill state multiplied by a constant factor. The peak power of the PV installation is $5\,\mathrm{kW}$. An immersion heater with $5\,\mathrm{kW}$ thermal power can be used to produce DHW if the heat pump is unable to fulfill all demands sufficiently.

We use a data set recorded for a real single family home in Düsseldorf during the year 2018. The data set consists of time series for outside temperature, PV generation, electrical demand, heat demand, and DHW demand with a resolution of 15 minutes per data point. The data set was obtained in an informal cooperation with Prof. Bernd Thomas from Reutlingen University.

2.1.2.2.2 Scheduling Algorithm We employed the MILP model from [SHM19] and modified it to compute heat pump schedules. To that end, we modelled heat and DHW as production goods that can be consumed in continuous units when requested by the residents. However, the heat pump uses electrical energy to generate heat and DHW, but the original model in [SHM19] does not allow electrical energy as production good. This is due to the nature of electrical energy which must be consumed at exactly the times it is produced. Additionally, electrical power is combined from multiple sources, i.e., the PV cells and the power grid, and the consumption from the grid should only be used for the residual power that cannot be served by the PV cell. However, the required modifications are minimal. A description of the model can be found in Appendix 2.1.

2.1.2.3 Results

We conducted an extensive case study with the model and data set from the single family home in Düsseldorf. The case study evaluated the required grid energy and on/off cycles per year under various conditions. We started with a very idealized model and successively added more realistic features and limitations. In this way, we were able to quantify the penalty of each of these features and limitations individually. All evaluations were conducted for heat pumps with and without power modulation as not all heat pumps have modulation capabilities. In the following, we will always give results for heat pumps without modulation first, followed by the respective result for heat pumps with modulation. All results obtained without forecasts are compiled in Table 2.2.

Variant	W/o modulation		W/ modulation	
	Grid energy (kWh)	On/off cycles	Grid energy (kWh)	On/off cycles
Entire year (min. grid energy)	≥ 4011	–	3897.47	600
Entire year (min. on/off cycles)	–	≥ 3	–	≥ 75
Rolling horizon (min. grid energy)	4078.91	1332	3901.59	1458
Rolling horizon (min. on/off cycles)	7975.60	256	7635.27	194
Rolling horizon/weighted sum, C=5 kWh	4478.52	239	4239.23	150

Table 2.2: Required grid energy and on/off cycles for optimized heat pump schedules with and without modulation. Adopted from [STM23].

We started by computing optimized schedules with the entire year as planning horizon and ground truth as forecast for future demands and PV generation. We conducted one optimization to estimate the minimum required grid energy, and another optimization to estimate the minimum required number of on/off cycles. While we computed the

minimum required grid energy to be 4011 kWh and 3897 kWh, we were not able to estimate the number of required on/off cycles due to computational limitations. However, optimizing an entire year at once is not reasonable in practice as demand and supply of energy cannot be known so far in advance. Thus, we employed the rolling horizon approach from [SHM19] with a planning horizon of 3 days.

Like the first optimization of an entire year, we used the rolling horizon approach once to minimize grid energy consumption, and once to minimize on/off cycles. The evaluation revealed that considering only a single objective yields poor results for the respective other objective. Partially, this can be explained by the fact that both objectives are contradictory. Reducing the number of on/off cycles per planning horizon results in the operation of the heat pump at times that are not beneficial. Similarly, minimizing the grid energy consumption results in frequent on/off cycles to only operate the heat pump at the most beneficial times. Therefore, both objectives must be balanced.

We employed weighted sum optimization to overcome this problem, i.e., we used a linear combination of both objectives with the rolling horizon method. We evaluated some weights and found a weight of 5 kWh per on/off cycle to be a good tradeoff between both objectives. Intuitively, this can be interpreted in the following way: an additional on/off cycle of the heat pump is only justified if at least 5 kWh grid energy consumption can be saved by allowing the additional on/off cycle. The weighted sum method resulted in less on/off cycles (239 and 150) than the rolling horizon approach with grid energy minimization (1332 and 1458), and less grid energy consumption (4478 and 4239 kWh) than the rolling horizon approach with on/off cycle minimization (7975 kWh and 7635 kWh). Thus, it is able to balance the short-term goal of minimizing grid energy consumption with the long-term goal of minimizing on/off cycles.

We used ground truth as forecasts for future PV generation and heat/DHW demand in all evaluations discussed so far. While forecasting PV generation can be reliably done with state of the art methods, forecasting heat and DHW demand for small-scale consumption is an open problem. Therefore, the results obtained so far were still not fully applicable in practice. We utilized a simplistic forecasting strategy to evaluate the costs of forecasting. The time series of the last day is repeated three times and used as forecast. This method is known as N-day-back in the forecasting literature. Despite its simplistic nature, it is known that this method yields good results. However, the usage of forecasts results in forecasting errors. For instance, the forecasting of less DHW demand than actually required results in a depletion of the DHW storage and thus demands that cannot be fulfilled. We overcame this problem during schedule execution

by moving unsatisfied demands in the next time slot and starting the heat pump immediately in case the state of charge of a storage falls under a certain threshold. The heat pump is operated until heat and DHW storage are completely filled, and finally, a new schedule is computed with a planning horizon that starts with full storages. Additionally, a new schedule is computed whenever the state of charge of a storage falls under a certain threshold. In this way, the schedule is updated for the actual system state.

We quantified the impact of the forecast methodology on grid consumption and on/off cycles for various thresholds. Larger thresholds resulted in less unsatisfied demand, but also in frequent reoptimizations and thus more required grid consumption and on/off cycles. While the penalty for additional purchased grid energy compared to ground truth was moderate (13.1% – 22.5%), significantly more on/off cycles are required (32% – 97.3%). This is due to the methodology of reoptimizations in case of forecasting errors. When a storage falls below the recharging threshold, an additional on/off cycle is introduced to refill the storages. Additionally, this run may be at a time with low PV power generation which explains the increase in grid consumption. We showed that this problem can partially be mitigated by installing a larger DHW storage.

2.1.2.4 Conclusion & Outlook

In this work, we presented a scheduling algorithm for computing multi-objective heat pump schedules. The contribution of this algorithm is the capability of minimizing a long-term objective with short-term schedules despite a conflicting short-term objective. We employed this algorithm to conduct an extensive case study for a residential building with roof-top PV cells. We quantified the costs imposed by forecasting energy production and consumption. We showed that even a simplistic forecasting strategy offers significant savings compared to a naive scheduling approach. Additionally, we discussed seemingly counterintuitive results regarding the usage of an immersion heater that resulted from the proposed methodology.

Future works may evaluation the potential for cost savings when a battery storage is integrated as an additional model component. A battery storage could be used to buffer overproduction of the PV cells for later consumption. However, overproduction of PV energy is only a matter during the summer, but heat and DHW demand are low at this time. Battery storages can only hold a few kWh, so the overproduction of the summer cannot be used during winter. Considering the acquisition costs of a battery storage, further evaluations are required to justify the integration of a battery storage.

The presented work is currently under review and can be found in Appendix 2.1.

2.1.3 Comparison of Forecasting Methods for Energy Demands in Single Family Homes

This section summarizes the research results from Stüber et al. [SHTM21]. This work is part of the additional content of this thesis. Therefore, it is only briefly summarized.

Summary

The computation of optimized heat pump schedules from Stüber et al. [STM23] (cf. Appendix 2.1) requires forecasts for heat and DHW demands. However, forecasting demands for single family homes is hard as they depend on the behavior of the residents which may change from day to day. We evaluate various simple time series forecasting strategies for heat and DHW in Stüber et al. [SHTM21]. These evaluations are based on the data that was used in Stüber et al. [STM23]. We compare the forecasting errors obtained from N-day-back methods, i.e., using the time series of the day N days in the past, with linear regression models. Additionally, we employ moving averages to smooth out random fluctuations, a common methodology in the forecasting literature. We show that despite its simplistic nature, the 1-day-back method yields acceptable results even without smoothing. These results motivated the use of the 1-day-back method in Stüber et al. [STM23].

2.1.4 Load Profile Negotiation for Compliance with Power Limits in Day-Ahead Planning

This section summarizes the research results from Heimgärtner et al. [HHS+19]. This work is part of the additional content of this thesis. Therefore, it is only briefly summarized.

Summary

In Stüber et al. [SHM19], we computed optimized production schedules for a cement plant. The electrical energy for the cement plant was obtained at the day-ahead market.

However, the day-ahead market is only for large scale trading, i.e., small and medium sized consumers cannot participate. Instead, they order their energy from an aggregator which buys large quantities of power. Unfortunately, most consumers do not want to share detailed information about their respective production process. For this reason, they deliver load profiles to the aggregator which describe the energy demand per hour. Not all demands can be satisfied due to power limits in the power grid. Thus, the consumer units provide multiple load profiles per day such that the aggregator can select one for each consumer. In Heimgärtner et al. [HHS+19], we propose two methods based on linear programming for load profile negotiation in such a case. The sequential method approves load profiles one after another. The simultaneous method approves load profiles for all consumers at once. The evaluation shows that the sequential approach results in slightly more consumers getting their favoured load profile approved than the simultaneous approach. However, the sequential approach can be considered unfair as the submission order of load profiles matters.

2.2 Algorithms for Resource Management in Real-Time Networks and Multicast Protocols

This section summarizes the research results of this thesis on management algorithms for communication networks. It covers five publications. All five publications are part of the core content of this thesis. First, the works about scheduling algorithms in TSN are discussed. Then, the publications about stateless multicast protocols are summarized.

2.2.1 Introduction to Time-Sensitive Networking

Real-time guarantees in communication networks are QoS guarantees that depend on the elapsed physical time between events in the network. For instance, the elapsed time between end-to-end transmission and reception of a frame is denoted as latency and bounded latency for frames may be a real-time guarantee. TSN is a set of standards that enhance bridged Ethernet networks for reliable data transmissions with real-time guarantees. The standards can roughly be categorized in standards for time synchronization, traffic shaping and scheduling, and network management. The following section gives a brief overview of traffic scheduling with TSN. Then, the scheduling problem of the

TAS is introduced. A comprehensive tutorial on TSN and the scheduling problem can be found in [SOLM23] and in Appendix 1.2.

2.2.1.1 Basics

TSN is an enhancement to bridged Ethernet networks [80291] with VLAN tagging [80218]. Devices that are sources of data transmissions are denoted as Talkers, and devices that are destinations of data streams are denoted as Listeners. The general term for devices that are Talker and/or Listener is end station. Bridges are network devices that are not the source or destination of data streams, but that relay frames between different LANs. The egress ports of a bridge may implement up to eight priority queues for waiting frames. These queues are first-in-first-out (FIFO), i.e., frames are served in arrival order. A so-called VLAN tag is added to the header of Ethernet frames. Among other information, the VLAN tag contains the Priority Code Point field. This 3 bit field is used to map frames to priority classes and thus priority queues in egress ports of bridges.

Many applications of real-time communication networks require network devices to have a common understanding of time to coordinate their actions. Thus, every bridge and end station is equipped with a clock. TSN features a protocol for clock synchronization denoted as generalized Precise Time Protocol (gPTP). The gPTP offers sub microsecond precision for networks with a diameter of at most 7 hops when implemented properly.

Traffic shaping techniques distribute frame transmissions in time. They are used to reduce buffering and congestion in a network due to traffic bursts. TSN offers various traffic shaping mechanisms, e.g., the so-called Credit-Based Shaper (CBS) in IEEE 802.1Qav [80210]. The enhancement for scheduled traffic defined in IEEE 802.1Qbv [80216] allows the implementation of the so-called Time-Aware Shaper (TAS) and is the most important feature of TSN for this thesis. Figure 2.5 depicts the components of the TAS. The enhancement adds a so-called transmission gate to each of the eight priority queues of an egress port. A transmission gate is in one of two states at any given time. These states are denoted as open and closed. Frames of a queue with a closed gate cannot be dispatched, i.e., only frames from queues with an open gate are eligible for transmission. The states of the eight transmission gates of an egress port are time controlled by a so-called Gate Control List (GCL). A GCL entry contains a time interval and a bitvector indicating the state of all eight gates of the respective egress port

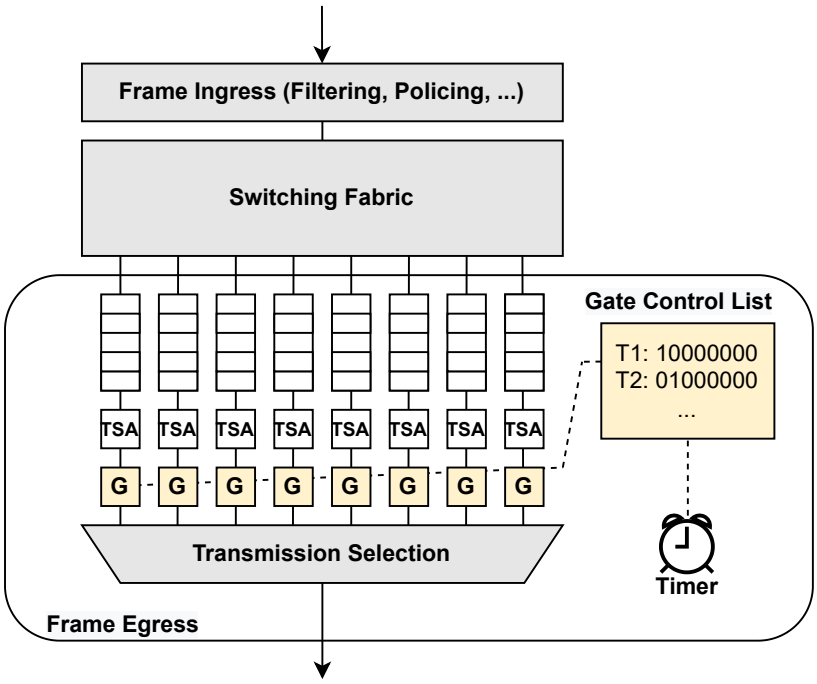


Figure 2.5: Processing pipeline of a bridge with the enhancement for scheduled traffic. Adopted from [SOLM23].

in the stated time interval. Thus, individual streams or traffic classes can be protected by not allowing other frame transmissions at the same time. GCL are periodic, i.e., the first entry is executed again after the time interval of the last GCL entry has passed.

By synchronizing all clocks in a network, selecting transmission offsets at end stations for frames, and computing appropriate GCLs for all bridges in a network, it is possible to guarantee per stream real-time requirements. The process of computing transmission offsets at end stations and GCLs for bridges is denoted as traffic scheduling. While IEEE 802.1Qbv defines the mechanism needed to implement traffic scheduling, no algorithm is specified to compute transmission schedules. The following section defines the traffic scheduling problem of the TAS.

2.2.1.2 The Scheduling Problem

The task of the scheduling problem is to find a traffic schedule for a given network topology and a set of data streams with real-time requirements.

2.2.1.2.1 Input The network topology is given as a graph $G = (V, E)$. The vertices in V correspond to the bridges and end stations of the network. The edges in E correspond to the links in the network. Links are full-duplex, i.e., they can be used for data transmissions in both directions at the same time. Bridges, end stations, and links may be equipped with additional properties such as propagation delays, processing delays, and transmission bandwidths. A data stream is the logical unit of one or more frames that transmit a message from an end station to one or more end stations. Thus, a data stream may be unicast or multicast. Data streams are periodic, i.e., they are repeated after some predefined and fixed time. This time is denoted as period. The description of a data stream features at least the period, the size of its frames, the number of frames per period, the source of the stream, and the set of receivers. Additionally, a deadline is typically given as real-time requirement. The deadline of a stream is the time all frames must have arrived at all receivers of the stream. The stream paths may be part of the input or subject of the scheduling problem. The literature considers various problem variations and modifications to the scheduling problem. The most important such variations are discussed in Section 2.2.2.2.

2.2.1.2.2 Remark About Periods Not all streams of a problem instance must have the same period. In this case, multiple repetitions of some or all streams must be considered to obtain a periodic schedule. The literature defines the so-called hyperperiod H to be the least common multiple of the periods of all streams in a problem instance. The hyperperiod is the period of the resulting traffic schedule. Thus, a stream with period p is represented by $\frac{H}{p}$ repetitions in the schedule. Deadlines are given relative to the start of a hyperperiod. Likewise, the times in the resulting schedule are stated relative to the start of a hyperperiod. An equivalent interpretation of this modelling approach is that a stream with period p is replicated $\frac{H}{p}$ times with each copy having period H. We will use this approach for simplicity and omit the handling of stream repetitions in this thesis.

2.2.1.2.3 Output A traffic schedule constitutes of transmission times of all frames of all data stream at their respective source node. Additionally, a schedule contains GCLs for all egress ports in the network topology. A schedule is considered valid if all frames arrive before their respective deadline when all end stations and bridges adhere to the transmission times and GCLs stated by the schedule.

2.2.2 A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN)

This section summarizes the contributions of the research work from Stüber et al. [SOLM23]. This publication is part of the core content of this thesis. First, the research objective is introduced. Then, the findings are summarized.

This work has been published as a journal paper in IEEE Access [SOLM23] (cf. Appendix 1.2) in 2023.

2.2.2.1 Research Objective

The enhancement for scheduled traffic IEEE 802.1Qbv [80216] introduced the capabilities for traffic scheduling in TSN. The traffic shaping mechanism that can be implemented with this enhancement is denoted as Time-Aware Shaper (TAS) (cf. Section 2.2.1). However, the standard only defines the details required to implement traffic scheduling from a technical point of view. An algorithm to compute traffic schedules is not specified in the standard. A large body of literature has developed since 2016 proposing algorithms for this purpose.

The goal of this work was to review the state of the art of scheduling algorithms for the TAS. We give a tutorial about TSN and common scheduling methodologies employed in the literature. Then, we develop a categorization of research works based on the considered problem variation. We compile important information, e.g., network topology, number of streams, and objective function from the reviewed works in tabular form. Finally, we present open problems not sufficiently covered in the literature so far.

We only present the categorization of research works and the open problems in this section. A detailed discussion of the reviewed works can be found in Stüber et al. [SOLM23] and Appendix 1.2.

2.2.2.2 Categorization

Many research works consider modified variants of the scheduling problem presented in Section 2.2.1.2. These problem variations add additional constraints, e.g., bounded number of available GCL entries, or degrees of freedom, e.g., the selection of stream paths. Additionally, the presented evaluations may focus on specific aspects, e.g., queuing delays. We classified research works based on these features. Figure 2.6 presents

the resulting categories and the reference numbers in the survey of research works in these categories. In the following, we introduce the most relevant categories.

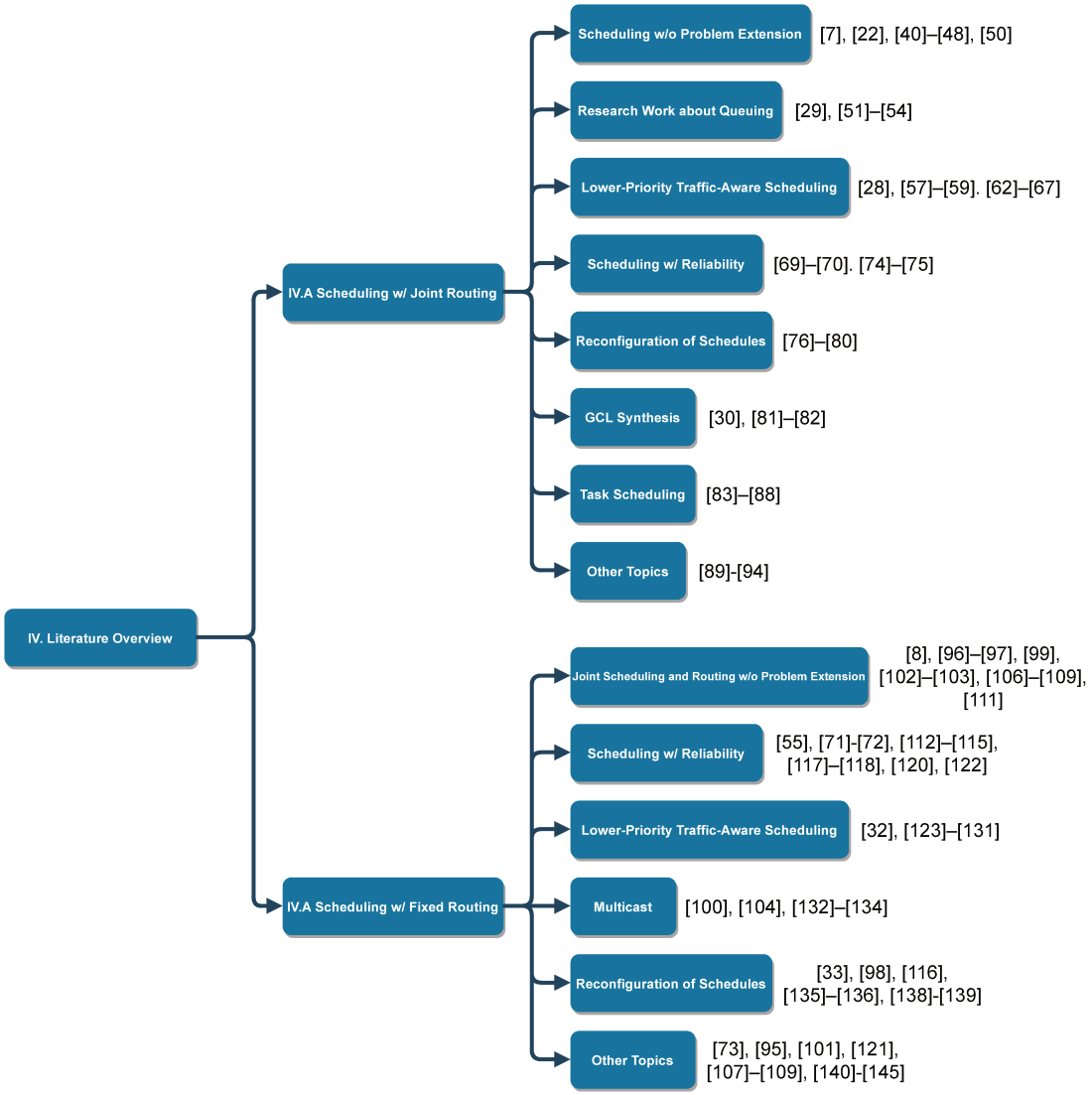


Figure 2.6: Organization of the Survey. The reference numbers refer to references in Stüber et al. [SOLM23] (cf. Appendix 1.2). Adopted from [SOLM23].

2.2.2.2.1 Joint Routing The unmodified scheduling problem assumes that all stream paths are given and are part of the input. However, there may be problem instances that cannot be scheduled due to some bottleneck link or complex dependencies between different streams. These conflicts can possibly be resolved when the scheduling algorithm is allowed to change stream paths. If the stream paths are an additional degree of freedom, the corresponding problem is denoted as the joint routing and scheduling problem. Thus, stream paths are part of the output of a joint routing and scheduling

approach.

2.2.2.2.2 Considerations for Other Traffic Classes There may be other traffic classes besides TT traffic in a TSN network. For instance, Audio Video Bridging (AVB) and Best Effort (BE) traffic can be combined with TT traffic. The quality of service (QoS) of these traffic classes depends on the schedule for TT traffic. Some research works integrate such considerations. For example, they introduce gaps between TT frames to allow the frequent transmission of AVB frames.

2.2.2.2.3 Queuing Queuing is a controversial topic in the TSN scheduling community. If multiple frames are scheduled to wait at the same time in the same queue, spontaneous frame loss or the failure of a Talker may lead to the transmission of another frame at the wrong time. For this reason, some works do not allow queuing delays at all. Other works do not allow frames of different streams to wait in the same queue at the same time. We classify research works that introduce novel ideas about the interplay of scheduling and queuing in this category.

2.2.2.2.4 Reliability Due to non-deterministic behavior of hardware devices, the execution of a schedule may fail, i.e., a frame may arrive after its deadline. For instance, spontaneous frame loss may occur with a small but not vanishing probability. Other examples for failures are bridges that drop valid frames, single link failures, and Talkers that do not sent scheduled frames due to a software fault. However, many use cases of TSN cannot tolerate any undesired behavior at all due to safety requirements. Some research works compute schedules that consider such fault scenarios. Most of the proposed approaches employ Frame Replication and Elimination for Reliability (FRER) [80217b]. Redundant copies of the same frame are sent via disjoint paths (1+1 protection). For instance, the computed schedules do not result in undesired behavior in case of a single link failure as at least one copy of the frame still arrives. However, if multiple frames arrive at the respective Listener, the redundant copies are filtered.

2.2.2.2.5 Dynamic Reconfiguration Not all use cases of TSN are static. Streams may be added or removed on the fly when the system is running. Some works propose methods to modify schedules without the need to reschedule all streams. Other works present novel ideas on how schedule updates can be deployed when the system is running without frame losses.

2.2.2.2.6 GCL Synthesis Most research works do not state explicitly how GCLs are constructed. Instead, they describe an algorithm that plans frame transmissions in time. GCLs are computed with a postprocessing in these works. However, the number of GCL entries is limited in hardware bridges and GCLs may be used to implement other features, e.g., the protection of different traffic classes from each other. We group research works in this category whether they construct GCLs explicitly or propose novel ideas about how GCL may be employed.

2.2.2.2.7 Task Scheduling Processes on Talkers and Listeners are denoted as tasks. These tasks run on a real-time operating system and produce and consume messages that are transmitted via TSN. Thus, the execution of tasks must be scheduled such that their schedule is consistent with the TSN schedule. Research works that schedule tasks and frames simultaneously are grouped in this category.

2.2.2.3 Shortcomings in the Literature

While reviewing the current state of the art in scheduling for the TAS, we found some shortcomings that applied to many published works. Most research works focus on the evaluation of runtimes of the proposed algorithms, i.e., experiments are conducted, and the wall-clock time needed to compute a schedule is measured. We acknowledge that scalability is an important property when an algorithm must be selected for a real-world use case. However, schedule properties, e.g., average latencies or GCL usage, are neglected in many of these works and they do not analyze details of high quality schedules. We also spotted questionable evaluation methodologies. For instance, some works report evaluation results individually per problem instance for a small number of problem instance (less than 10). Thus, reported results are more of anecdotal character and create the impression of cherry picking to support a specific conclusion. Additionally, some works do not describe problem instances sufficiently, e.g., important properties such as the network topology are missing.

Another shortcoming in many works is the lack of comparison to similar works. Most authors compare their new algorithms only to their own algorithms from previous years and conclude that their new algorithm is even better than their old one. Problem instances used for evaluation are not published such that results cannot be reproduced. Therefore, it is hard to compare research works from different authors. We tried to over-

come this shortcoming by evaluating 11 well-known algorithms on a publicly available set of problem instances in [SEOM24] and Appendix 1.3.

Another subtle shortcoming in the literature is the use of non-standard vocabulary. TSN is an extension to bridged Ethernet networks and uses the same specific jargon in its standards. Additionally, TSN is a layer 2 technology, not a layer 3 technology. Thus, the correct terms are *frame*, *stream*, and *path selection*, not *packet*, *flow*, and *routing*. Devices that relay frames are denoted as *bridges*, sources and destinations of data streams are *end stations*.

2.2.2.4 Open Problems

We concluded the survey with a discussion of open problems. More elaboration on these problems can be found in [SOLM23, Section VII.B]. While some works consider scheduling in the presence of non-scheduled traffic such as AVB and BE traffic, it is still a major open problem to combine TAS scheduling with multiple other traffic classes simultaneously. Additionally, different traffic classes require the use of guard bands and their effect is mostly ignored in the literature so far. There are many works that consider the joint routing problem, but a scalable exact solving approach was not developed so far. Although few works allow multicast streams, no evaluation results about the impact of multicast streams are reported in the literature.

Current works restrict the usage of queuing to prevent certain results of non-deterministic behavior. However, these restrictions are not mandatory with respect to the TSN standards and the performance impact compared to unrestricted queuing is unknown. Other causes of non-determinism are completely ignored, e.g., jitter in processing and propagation delays. TSN offers a standard denoted as Per-Stream Filtering and Policing (PSFP) in IEEE 801.1Qci [80217a] that allows to protect a schedule from unexpected frames. However, there is currently no joint approach for TAS scheduling and PSFP available. Additionally, PSFP may be used to implement security features such as protection against Denial-of-Service (DoS) attacks in the future.

TSN is a rather new technology for layer 2 communication in industrial applications. Typically, these applications are designed for years or decades and almost all devices deployed today are still not capable for TSN. Thus, it is an important open problem to integrate legacy devices in TSN networks as such devices will be around for quite some time.

Finally, the major open problem in scheduling for the TAS is to develop a single approach that covers all features offered by TSN. TSN consists of more than 10 standards which can be used together in theory. Unfortunately, all algorithms in the literature cover only a small subset of these standards, often none except for the enhancement for scheduled traffic [80216]. For instance, there is no research work that allows to use more than 2 different traffic shapers simultaneously, but TSN offers at least 5 different shapers at the time of writing this thesis. The long-term goal for future approaches should be to integrate more or even all standards into a converged approach to enable users to exploit the full potential of TSN.

2.2.2.5 Conclusion

In this work, we presented a rigorous overview of the state of the art in computing traffic schedules for the TAS. We presented a self-contained tutorial about various TSN standards, and commonly used scheduling methods from the literature. We summarized important findings and compiled common features of the surveyed works extensively. Additionally, we proposed suggestions for improvements for future works and highlighted open problems not discussed in the literature so far.

A potential shortcoming of the presented work is the lack of comparing evaluations. However, we have conducted a survey and not a systematic review which implies that evaluations are not in scope. Unfortunately, the literature lacks an comparison study that evaluates the advantages and shortcomings of state of the art scheduling methods. Most authors do not compare their algorithm to other works or only to algorithms developed by themselves before. However, we conducted research to fill this gap in [SEOM24] which is presented in the next section.

2.2.3 Performance Comparison of Scheduling Algorithms for Time-Sensitive Networking (TSN)

This section summarizes the contributions of the research work from Stüber et al. [SEOM24]. This publication is part of the core content of this thesis. First, the research objective is introduced. Then, the data set is explained and an overview of the compared algorithms is given. Finally, the results of the case study are summarized.

This work was accepted for publication in the journal IEEE Transactions on Industrial Informatics (TII) and can be found in Appendix 1.3.

2.2.3.1 Problem Description

The survey of Stüber et al. [SOLM23] revealed that the literature proposes more than 100 scheduling algorithms for the TAS. These algorithms differ in the employed solving methodology, e.g., ILP solving, genetic algorithms, or Tabu search, and the modelling assumptions, e.g., whether queuing delays are allowed or not. However, most of the algorithms were constructed to solve the same problem, i.e., finding a valid schedule for a set of streams with real-time requirements with few or no additional constraints. Most of the respective research works omit quantitative comparisons with algorithms from other authors. For these reasons, it is hard for engineers and other practitioners to select an algorithm for a specific use case. Thus, the first research objective of this work is to conduct insightful parameter studies comparing well-known algorithms and to give recommendations based on the results. We selected 11 seminal works and implemented their algorithms according to the respective paper for these comparison studies.

Another issue in the community of TAS scheduling regards the problem instances used for evaluations. Almost all research works do not publish their problem instances. Thus, evaluation results cannot be reproduced or validated. To make matters worse, many works do not describe their problem instances sufficiently. The descriptions lack important parameters, do not describe the network topology, or do not discuss how streams were sampled. Therefore, the second research objective of this work is the scientifically rigorous construction of a set of problem instances that can be used by future works. This set should cover problem instances for various parameter studies and the problem instances must represent real-world use cases of TSN, e.g., automotive or factory automation use cases. The set is released to the public to allow the validation of the evaluation results by independent researchers. Additionally, authors of future algorithms can use the reported results on these instances as baseline in a comparison with their new approaches.

2.2.3.2 Concept

We describe the sampling of the problem instances and give a brief overview of the compared algorithms.

2.2.3.2.1 Problem Instances A problem instance for the scheduling problem constitutes of a full description of all data streams and the network topology needed as input

to compute a schedule. A parameter is a property that guides the sampling of problem instances. Parameters can be deterministic, e.g., the number of frames, or random, e.g., the distribution of stream periods. We constructed sets of problem instances for various parameter studies. All problem instances in a parameter study were sampled with the same set of parameters except for the studied parameter. In this way it is possible to observe the effect of changing the studied parameter in isolation. We constructed 20 problem instances per parameter configuration per parameter study. Thus, it is possible to calculate confidence intervals for evaluation results. Each parameter study was constructed two times, once for sparse topologies and once for highly meshed topologies. This allows to capture the different behavior of scheduling algorithms in these cases.

Random parameters may result in noise. For instance, frame sizes and stream periods are random parameters that control the overall traffic volume during a hyperperiod. It is undesirable that this affects evaluation results in a parameter study for the number of bridges in a network topology. We sampled 40 default realizations for all random parameters to overcome this issue. These default realizations were used whenever applicable. In terms of the previous example, 40 default realizations for frame sizes and stream periods were constructed and reused in the problem instances of all numbers of bridges. Thus, the effect of random noise is eliminated when comparing results for different numbers of bridges.

The possible values of the parameters were chosen to resemble actual use cases of TSN and the current state of the art of scheduling algorithms. For instance, the set contains problem instances with stream periods that represent automotive [KZH15], avionics [BSN+14], and factory automation [Ind18] use cases.

We sampled an additional set of infeasible problem instances with 100 streams per instance. Each of these problem instances contains a set of 10 streams such that the respective problem instance is infeasible with these streams but becomes feasible when any of the 10 streams is removed.

Table 2.3 compiles the parameters and their possible values used to construct parameter studies. We remark that two topologies are denoted as default values for the topology parameter. Random regular graphs (RRGs) are the default parameter for parameter studies in highly meshed topologies. Ring topologies are the counterpart for sparse topologies.

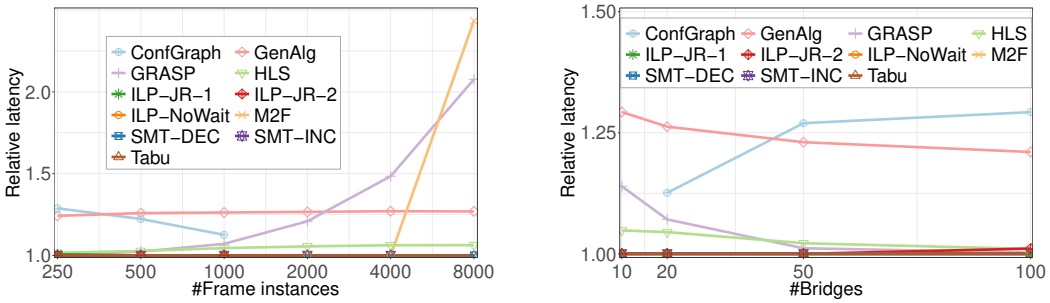
Parameter	Possible values
#Bridges	10, **20***, 50, 100
#Frame instances	250, 500, **1000***, 2000, 4000, 8000
Topology	Line, **ring***, star, **RRG***, scale-free
#Frames/period	**1***, 2, 4
Frame size f_s	**Random***, $84\,B$, $813\,B$, $1542\,B$
Stream periods T_s	$\{0.5\,\text{ms}\}$, $\{1\,\text{ms}\}$, $\{\mathbf{1\,ms,\ 2\,ms}\}^*$, $\{20\,\text{ms}, 50\,\text{ms}, 100\,\text{ms}\}$, $\{2\,\text{ms}, 16\,\text{ms}, 128\,\text{ms}\}$
Latency	0.25, 0.5, **1*** $\times$ the respective stream's period
#Listeners/stream	**1*** (unicast), 2, 4, 8, 16

Table 2.3: Parameters and their possible values. Default values are **bold** and indicated by "*". Table adopted from [SEOM24].

2.2.3.2.2 Algorithm Selection We selected the algorithms such that every important solving methodology was represented by at least one algorithm. Additionally, we preferred seminal works from the literature which influenced many recent works. Dürr et al. [DN16] proposed an ILP (ILP-NoWait) and a Tabu heuristic (Tabu) for no-wait scheduling. In contrast, Craciunas et al. [COCS16] presented an SMT-based approach (SMT-INC) that allows frame queuing. This approach was enhanced for scalability by Pozo et al. [PSRNH15] (SMT-DEC). Gavrilut et al. [GZRP18] and Jin et al. [JXG+20] proposed problem specific heuristics (GRASP and M2F) that allow frame queuing. Up to this point, all algorithms assumed stream paths as fixed prior to scheduling.

A genetic algorithm (GenAlg) and a list scheduler (HLS) for no-wait scheduling with joint routing were presented by Pahlevan et al. in [PO18] and [PTO19]. Falk et al. [FDR18] proposed an ILP approach (ILP-JR-1) to compute paths and no-wait schedules simultaneously. Schweissguth et al. [STP+20] presented a very similar ILP model (ILP-JR-2) with enhancements for scalability. Finally, Falk et al. [FDR20] developed a problem specific heuristic (ConfGraph) based on the computation of independent sets in conflict graphs.

The algorithms employ different solving methodologies, differ on the facts whether frame queuing is supported, and whether the routing is considered as fixed prior to scheduling. Thus, the reader might suspect that a comparison of these algorithms is not fair. However, all algorithms were designed to solve the same problem. It is the authors' opinion that different design decisions compared to other algorithms are not a fairness issue. If a methodology A can compute results with higher quality than another methodology B in the same time or even faster, this is an advantage of A over B.



(a) Varying number of frames sent from Talkers.

(b) Varying number of bridges.

Figure 2.7: Relative stream latencies of the compared algorithms for different numbers of frames and bridges per problem instance. Figures from [SEOM24].

2.2.3.3 Results

We conducted various parameter studies with the problem instances discussed in Section 2.2.3.2.1 and the algorithms from Section 2.2.3.2.2. We reported comparison results about solving time, scalability, multi-threading speedup, GCL length, scheduling under challenging conditions, and stream latencies. In the following, we discuss the stream latency evaluation as an example and refer to Section VI of Stüber et al. [SEOM24] and Appendix 1.3 for the other evaluations.

The average stream latency is an important quality measure for schedules as most use cases of TSN require ultra-low latencies. However, latencies of problem instances with different topologies are hard to compare. Thus, we introduced relative stream latencies to overcome this problem. To that end, we calculated stream latencies relative to the minimum required path delay of a stream, i.e., the sum of processing, propagation, and transmission delays along the stream's path. We conducted two evaluations. First, we varied the number of frames sent from Talkers, i.e., we computed schedules for 250, 500, ..., 8000 frames sent per hyperperiod from all Talkers combined. We used network topologies with 20 bridges in this evaluation. Second, we varied the number of bridges in a network topology, i.e., the size of the considered network. The number of sent frames per hyperperiod was 1000 in this evaluation. Figures 2.7a-2.7b depict the relative stream latencies of the computed schedules. The evaluations revealed significant differences between the scheduling algorithms compared. Algorithms that resolve resource conflicts by introducing queuing delays (M2F and GRASP) resulted in up to three times higher stream latencies than necessary. Joint routing heuristics (GenAlg, HLS, ConfGraph) yield schedules with high latencies due to longer stream paths. Tabu,

SMT-DEC, SMT-INC, ILP-NoWait, and ILP-JR-{1,2} produced schedules with minimum latencies.

Based on all evaluation results, we gave recommendations about which algorithm to implement in different scenarios. All the evaluated algorithms impose some drawbacks. ConfGraph, GenAlg, and ILP-JR-{1, 2} scale extremely badly in all evaluations and should not be used in practice. GCL entries are a limited resource in most hardware implementation of the enhancement for scheduled traffic [80216]. Algorithms that result in small GCLs are beneficial if such bridges are used. Thus, we recommend M2F and SMT-DEC in those cases. ILP-NoWait and Tabu resulted in minimum latencies and are preferable compared to joint routing approaches (HLS, GenAlg, ConfGraph, ILP-JR-{1,2}) when ultra-low latencies are required. In case of frequent recomputations of schedules, Tabu, HLS, and GRASP should be used as they find first valid solutions fast. Finally, implementation costs may be significant for some of the algorithms. ILP solvers are expensive and may not be available for many practitioners and some of the algorithms are not sufficiently described in the respective research work. Thus, we recommend Tabu, HLS, SMT-DEC, and SMT-INC when costs are a limiting factor as they can be implemented with little effort.

2.2.3.4 Conclusion & Outlook

We proposed a set of problem instances that can be used as a reference for future researchers. The authors of future works can use these instances for evaluation to ensure reproducibility of their results. Additionally, we evaluated 11 seminal algorithms with various methodologies on these instances. The contribution of this comparison is twofold. First, the literature gap for a quantitative comparison of state of the art algorithms is filled. Second, future authors can compare their results without effort to existing approaches to assess the quality of their research.

A potential drawback of the presented work is the small number of metrics used to compare schedules. However, the used metrics are just the lowest common denominator of the algorithms proposed in the literature. It is not reasonable to compare problem variation specific objectives of algorithms that do not consider the respective problem variation. For instance, comparing the ability of Tabu and ILP-NoWait to compute schedules robust against frame loss is pointless as their objective functions do not consider this goal and results will be random.

Future works may propose problem instances for reliability and online reconfiguration settings. An evaluation and benchmark framework which could be the base of future algorithms would be the next step towards reproducibility and comparability in the field of TAS scheduling.

2.2.4 Efficient Robust Schedules (ERS) for Time-Sensitive Networking

The section summarizes the contributions of the research work from Stüber et al. [SOM24]. This publication is part of the core content of this thesis. First, the research objective is introduced. Then, the fault model and the scheduling algorithm are presented. Additionally, the proposed solution is briefly compared to state of the art approaches from literature. Finally, the findings are summarized.

This work is accepted for publication in a future issue of the IEEE Open Journal of the Communications Society journal and can be found in Appendix 2.2.

2.2.4.1 Problem Description

The enhancement for scheduled traffic [80216] allows to implement traffic scheduling in TSN (cf. Section 2.2.1.2). A common assumption in the literature is that all devices in a TSN network behave completely deterministically. The times of all clocks are perfectly synchronized, processing delays in bridges have equal duration for every frame, frames are never lost, and race conditions cannot happen for simultaneously arriving frames. All these assumptions may fail in practice. Disregarding these potential causes of non-deterministic behavior may result in lost frames and missed deadlines. However, many use cases of TSN, such as industrial automation and in-vehicle communication, are safety critical and cannot tolerate any non-determinism.

In addition to the mentioned non-deterministic behavior, most works in the literature make more assumptions that do not hold in practice. The number of GCL entries per egress port is limited in hardware bridges. Thus, schedules that open and close a gate too often cannot be deployed on hardware bridges. Even more problems arise in the presence of other traffic classes than TT traffic. For instance, assume that BE traffic may be transmitted during times that are not reserved for TT traffic. If a BE frame is available for transmission in an egress port and the remaining time before the gate for BE traffic closes is not sufficient to transmit the frame, the frame cannot be dispatched and blocks its queue. The time interval before a gate closing event in which such a

transmission conflict may happen is denoted as guard band. In the worst case, the transmission bandwidth during a guard band cannot be used. Thus, frequent gate events and small time slots for non-scheduled traffic classes may waste bandwidth.

The goal of this work was to develop an algorithm that computes traffic schedules that are robust against the discussed causes of non-determinism. That means that all frames arrive before their respective deadline even in case of non-deterministic behavior. The number of GCL entries used per egress port should be limited. For most problem instances, there are many valid schedules which respect all real-time requirements. The algorithm should maximize the remaining bandwidth for other traffic classes, i.e., the bandwidth blocked by guard bands or scheduled traffic should be minimized.

2.2.4.2 Concept

We identified four causes of non-deterministic behavior that must be considered for robust schedules. Based on the countermeasures for these causes, we derived a MILP model to compute optimized schedules. The schedules are optimized for maximizing the available bandwidth for non-scheduled traffic. This section introduces the fault model and the countermeasures against non-deterministic behavior.

2.2.4.2.1 Fault Model The duration of processing delays in bridges is subject to jitter due to physical effects. This may result in frames arriving earlier or later than scheduled in an egress queue. Let σ be the maximum possible deviation from the average processing delay d_{proc}. Thus, the range of possible processing delays is $[d_{proc} - \sigma, d_{proc} + \sigma]$. We consider processing delays to be sampled uniformly from this range.

So-called race conditions are another potential fault caused by hardware limitations. If two frames are scheduled to arrive almost simultaneously on different ingress ports of a bridge, and both frames are put in the same egress queue, the processing order may be non-deterministic. This may result in missed deadlines and wrong transmission times for other frames. We denote the minimum inter-arrival times of two frames that are forwarded by the same egress port such that their order in the egress queue is deterministic by λ.

Time synchronization errors are caused by not perfectly synchronized clocks in network devices. Assume device A sends a frame to a neighboring device B. Let T_A and T_b be the clock times of A and B, respectively. If $T_A < T_B$ and a frame is scheduled to be sent

at some specific time from A to B, the frame would arrive earlier than scheduled from the perspective of B relative to B's local time. Vice versa, if $T_B < T_A$, the frame would arrive later than scheduled at B. In the presence of GCLs, this may result in frames getting locked and failed transmissions due to closed gates. We denote the maximum tolerated time synchronization error with δ.

Finally, spontaneous frame losses can cause non-deterministic schedule deviations. Assume two frames are scheduled to wait some time in the same egress queue with a closed gate. If the first frame was lost at a previous hop of its path, the second frame is transmitted immediately at the time the gate is opened. Thus, this frame arrives earlier than scheduled at its next hop and may delay other frames, ultimately resulting in deadline misses in the worst case.

2.2.4.2.2 Countermeasures To compensate for processing jitter, we consider earliest and latest transmission times for every frame at every hop. All constraints of the scheduling problem, e.g., that frames must arrive before their respective deadline, must hold for the earliest and latest possible transmission and arrival times. Additionally, the countermeasures against time synchronization errors and race conditions must consider earliest and latest transmission times as well. The earliest and latest transmission times are calculated by assuming a minimum or maximum possible processing delay occurs at every hop during schedule execution, respectively.

The other countermeasures tackle multiple causes of non-determinism simultaneously. For instance, frames must arrive before their deadline even in the latest possible case. Additionally, frames must arrive δ before their deadline to compensate for the maximum time synchronization error. Similarly, the transmissions of two frames over the same link must not collide, even if one frame is sent at its earliest time, the other is sent at its latest time, and in the presence of a maximum time synchronization error.

To counteract race conditions even in the case of time synchronization errors, frame arrivals from different ingress ports must be separated by $\lambda + \delta$. Finally, the effects of spontaneous frame losses can be limited to not affecting other frames by enforcing so-called isolation constraints [COCS16]. These constraints enforce that only a single frame can wait in an egress queue at any given time. Additionally, these constraints must also hold in case of processing jitter and time synchronization errors. Thus, the latest possible transmission time of a frame must be separated by at least δ from the earliest possible arrival time of the next frame in the respective egress queue.

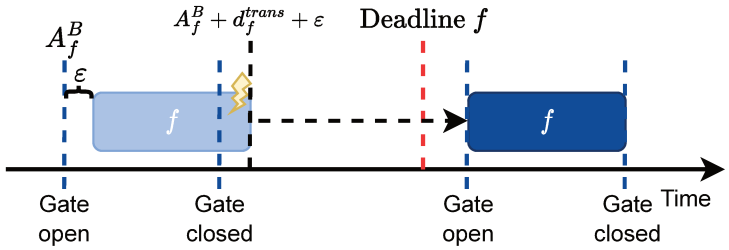


Figure 2.8: A frame arrives ε later than scheduled at an egress port. Transmission fails as the gate will be closed before transmission is finished. Adopted from [SOM24].

2.2.4.2.3 Scheduling Algorithms We derived a MILP model from the above discussion. The details of this model, i.e., its idea, the variables, and the constraints can be found in Section VII of Stüber et al. [SOM24] and Appendix 2.2. However, we highlight one difference between the proposed modelling approach and other works that limit the number of GCL entries [JXG+20][OCS18]. The presented modelling approach has the benefit that the number of variables does not depend on the maximum allowed number of GCL entries. We will use this fact to explain the performance gain compared to other approaches.

Craciunas et al. [COCS16] solves the discussed causes of non-determinism by introducing large gaps between frame transmissions. However, the necessary gaps are overestimated and result in the waste of bandwidth. Additionally, gaps between consecutive frames are not sufficient to resolve time synchronization errors. Figure 2.8 depicts an example where a frame f arrives later than expected at time $A_f^B + \varepsilon$ instead of A_f^B at some node B. If the gate closing are not scheduled to compensate for such delays, the remaining time before the gate closes is not sufficient for transmission and the frame may miss its deadline.

The proposed algorithm in this work has the advantage that gaps have the minimum required length to guarantee deterministic behavior. This cannot be done in advance as the minimum gap between frame transmissions depends on the schedule of other frames and the GCLs. For instance, the jitter of a frame can shrink along the frame's path if the frame must wait at closed gates in the earliest and latest possible case. We denote robust schedules computed by the proposed method as Efficient Robust Schedules (ERS). The solution proposed by the related work is denoted as Naive Robust Schedules (NRS). Schedules without any robustness considerations, i.e., without any gaps between frame transmissions, are denoted as Tight Schedules (TS). NRS and TS can be computed by a slightly modified version of the proposed algorithm.

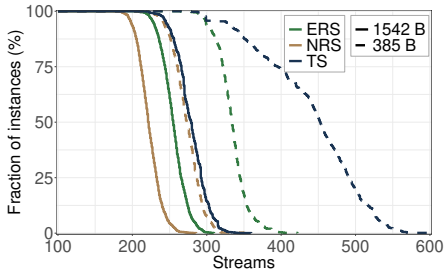


Figure 2.9: Fraction of admitted streams for ERS, NRS, and TS. Figure from [SOM24].

The required modifications to the MILP to compute NRS and TS can be found in the appendix of [SOM24].

2.2.4.3 Results

We conducted a case study to demonstrate the benefits of ERS over NRS. The comparison with TS quantifies the costs of robustness. All evaluations were conducted with ring topologies, which are common in factory automation use cases [HGF+20], and randomly generated streams. We assume bounds for time synchronization errors of $\delta = 1\,\mu s$, race conditions $\lambda = 0.4\,\mu s$, and processing jitter $0.15\,\mu s$. These values were derived from expert knowledge and discussions with a bridge manufacturer.

First, we evaluated the fraction of transmission capacity actually used for scheduled traffic with respect to the bandwidth reserved for scheduled traffic. This metric is not known in the literature so far and we denoted it as schedule density. We observed a significant benefit of ERS compared to NRS for varying numbers of GCL entries per egress port. This is due to the waste of bandwidth between frame transmissions. Gaps between transmissions are larger than necessary with NRS. TS results in even less wasted bandwidth, but these schedules may fail in practice. The difference to TS can be considered as the cost of robustness. We consistently observed the same result for varying frame sizes, processing jitters, and time synchronization errors.

The stream periods used in the evaluation ranged from $500\,\mu s$ to $1500\,\mu s$. Despite the seemingly small scale of time synchronization errors ($1\,\mu s$), race conditions ($0.4\,\mu s$), and processing jitter ($0.15\,\mu s$) compared to stream periods, we observed a significant impact to the number of admissible streams. For this evaluation, we added streams successively to a problem instance and checked schedulability with ERS, NRS, and TS. The experiment was repeated 20000 times for maximum sized frames, i.e., 1542 B, and for medium-sized frames, i.e., 385 B. Figure 2.9 depicts the CCDFs of admissible

streams for both frame sizes. We observed that significantly more streams can be admitted with ERS compared to NRS. While the difference between ERS, NRS, and TS is moderate for medium-sized frames, it is significant for large frames.

# streams	ERS	A1 [COCS16]	A2 [dSSN19]	A3 [OCS18]	A4 [JXG+20]
10	0.0136	0.0523	5.16	3.32	3600
20	0.027	0.15	28.6	23.7	3600
50	0.152	0.963	414	598	3600
100	0.902	8.57	647	3600	3600
200	24	438	1204	3600	3600
# bridges	ERS	A1 [COCS16]	A2 [dSSN19]	A3 [OCS18]	A4 [JXG+20]
5	0.537	5.1	527	3600	3600
10	0.902	8.57	647	3600	3600
15	1.26	14.1	752	3600	3600
20	1.87	23	903	3600	3600
Topology	ERS	A1 [COCS16]	A2 [dSSN19]	A3 [OCS18]	A4 [JXG+20]
Ring	0.902	8.57	647	3600	3600
Line	3.35	28.1	1461	3600	3600
Star	0.445	3.73	118	3186	3600
Full-mesh	0.109	1.56	6.04	316	3600
Scale-free	0.627	4.94	409	3022	3600

Table 2.4: Computation times in seconds for different TSN scheduling algorithms. Timeout was set to 3600 s. Table adapted from [SOM24].

Finally, we compared the proposed approach to state of the art algorithms. The algorithms A1 [COCS16], A3 [OCS18], and A4 [JXG+20] feature countermeasures against time synchronization errors and are essentially equivalent to NRS. The algorithms A2 [dSSN19], A3, and A4 restrict the GCL length to a predefined maximum length. We measured the time needed to find an optimized schedule for various network sizes, topologies, and stream numbers. Table 2.4 shows clearly the advantage of ERS in all these evaluations compared to state of the art algorithms. The proposed algorithm is at least 10 times faster than approaches without GCL synthesis (A1), and 50 times faster than approaches with GCL synthesis (A2, A3, A4).

2.2.4.4 Conclusion & Outlook

In this work, we identified causes of non-deterministic behavior in TSN. We showed that the solution approach from the literature for these problems results in a waste of resources. To that end, we proposed a scheduling algorithm that computes schedules robust against the discussed causes of non-determinism. We compared the approach from the related work with the new algorithm in various parameter studies. All these studies demonstrated that the presented approach is superior. Additionally, we showed that the problem is not neglectable and results in a significant waste of resources when

handled in a naive way. Thus, we conclude that the presented algorithm is currently the best available scheduling algorithm for TSN in presence of non-determinism. Additionally, it is the only algorithm in the literature that maximizes the available bandwidth for non-scheduled traffic.

Future works may investigate the effects of schedule deviations that result from non-deterministic behavior. For instance, a spontaneous frame loss may imply that other frames are sent earlier than scheduled. These frames may block other egress ports due to reordering. In the worst case, some other frame may miss its deadline and safety critical applications do not receive the required data in time. Preliminary results indicate that even small effects can actually result in deadline misses and heavy congestion. Thus, the contribution of this work is important and motivates future research works.

2.2.5 Introduction to Bit Indexed Explicit Replication

Multicast is the concept of sending a data stream from one source to multiple receivers. Today, IP multicast (IPMC) is the most common protocol for multicast traffic on the Internet. It allows a node to send an IP packet to multiple receivers along a tree such that at most one copy of the packet is forwarded per link on the tree. However, IPMC requires to maintain state in core nodes and the configuration of multicast groups results in signaling overhead. The IETF introduced BIER [WRD+17][MMWE18] to overcome these drawbacks of IPMC. Instead of storing information about multicast groups in core nodes, the receivers of a packet are explicitly encoded in a bitstring in the packet's header. Every possible receiver in a BIER domain is represented by one bit position in the bitstring. If and only if a bit is set to 1, the packet is forwarded and delivered to the respective receiver. Thus, the state in core nodes does not need to change if a receiver joins or leaves a multicast group.

Figure 2.10 illustrates the architecture of BIER. BIER is an overlay network over IP. That means that the IP underlay is used to determine packet paths and to forward BIER packets. The sender of a message sends an IPMC packet to a Bit Forwarding Ingress Router (BFIR). The BFIR encapsulates the IPMC packet with a BIER header. The bit positions in the BIER header's bitstring correspond to so-called Bit Forwarding Egress Routers (BFERs) of the BIER domain. The routers in the BIER domain forward the BIER packet to all BFERs with a 1 set at the respective position in the bitstring. Finally, the BFERs decapsulate the IPMC packet from the BIER packet and forward the packet via IPMC to its receivers.

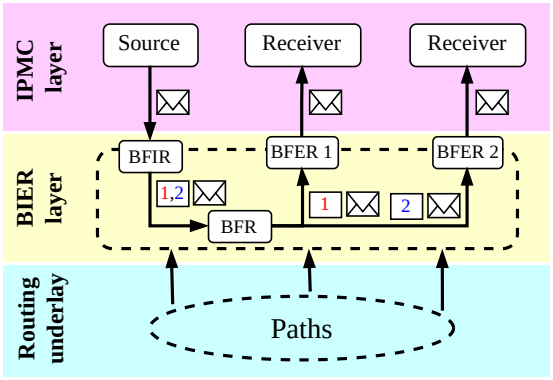


Figure 2.10: Layered architecture of BIER. Adopted from [MSM23].

2.2.6 Efficiency of BIER Multicast in Large Networks

The section summarizes the contributions of the research work from Merlin et al. [MSM23]. This publication is part of the core content of this thesis. We remark that the first two authors contributed equally to this publication. First, the research objective is introduced. Then, the traffic model and the heuristic algorithm are presented. Finally, the findings are summarized.

This work has been published as a journal paper in IEEE Transactions on Network and Service Management [MSM23] (cf. Appendix 1.4) in 2023.

2.2.6.1 Problem Description

BIER is a novel protocol for stateless multicast traffic that encodes packet receivers in a bitstring in the packet's header (cf. Section 2.2.5). Due to technical restrictions in forwarding nodes, the maximum size of a packet's header that can be processed is limited. Thus, a large BIER domain must be divided into so-called subdomains (SDs) as the bitstring would be too long otherwise. To that end, the BIER header features a Subdomain Identifier (SDI) field. The SDI determines the mapping of bitstring positions to BFERs. Thus, only the combination of SDI and bitstring controls which receivers are addressed by a BIER packet. A consequence of the implementation with SDIs is that a single BIER packet can only address BFERs of the same SD. If the receivers of a message are contained in multiple SDs, a BIER packet must be sent to every SD that contains at least one receiver. Thus, multiple BIER packets may be needed although a single IPMC packet would be sufficient. SDs are static, i.e., they are configured once and are not changed in favor of individual messages or multicast groups.

Not every partitioning of BFERs in SDs is equivalent with respect to various performance metrics. For instance, larger SDs result in less packets that are sent from a source node, but the header size of these packets is larger due to the longer bitstring. Thus, the sum of all data transmissions in the network needed to deliver the packet may increase even though fewer packets are sent. This example illustrates that there are non-trivial tradeoffs when assigning nodes to SDs.

The goal of this work is twofold. We develop an algorithm that partitions a BIER domain into SDs. To that end, we define a performance metric for this new problem that captures the amount of traffic transmitted in the network. We compare this algorithm to optimal solutions obtained by an ILP. Due to the near-optimal results of the algorithm for small topologies we evaluate the scalability of the BIER mechanism with the heuristic algorithm also in large networks. We compare hop counts, traffic amounts, link loads, and the impact of single link failures of BIER in large networks with IP unicast and IPMC.

2.2.6.2 Concept

We proposed a simple yet effective heuristic to partition a BIER domain into SDs. These SDs should be constructed such that the number of individual packet hops is minimized when all nodes send a BIER packet to all other nodes. Additionally, we presented an ILP approach for the sake of comparing the heuristic to optimal solutions in small networks. Even though the ILP does not scale for large problem instances, results obtained by it indicate that the heuristic produces near-optimal results. Thus, we leverage the heuristic to evaluate the scalability of the BIER mechanism itself.

2.2.6.2.1 Traffic Model and Performance Metric We assume that every node of a large BIER domain sends a multicast message to all other nodes of the BIER domain. We use the number of packet transmissions as performance metric, i.e., we sum up the number of packets carried over all links of the BIER domain. Let $\mathcal{V}$ be the set of nodes of some BIER domain and $\mathcal{P}$ be a partitioning of $\mathcal{V}$, i.e., a set of subsets of $\mathcal{V}$ such that every node is contained in exactly one subset. Furthermore, let $p(v, w)$ denote the shortest path between v and w as ordered set of links. The performance metric ρ is formally

defined by the following equation:

$$\rho := \sum_{v \in \mathcal{V}} \sum_{\mathcal{S} \in \mathcal{P}} \left| \bigcup_{w \in \mathcal{S}} p(v, w) \right| . \tag{2.1}$$

2.2.6.2.2 The Heuristic Algorithm We designed a heuristic that partitions a BIER domain into SDs such that the performance metric stated in Equation (2.1) is minimized. First, initial solutions are constructed by sampling a seed node for every SD followed by a heuristic denoted as bubble growing [DPSW00]. Essentially, bubble growing is a breadth first search that is started at every seed node and nodes are assigned to SDs in a round robin like manner. Then, the best initial solution found is used as starting point of a greedy hill climbing heuristic [RN10]. Nodes of different SDs swap their assignment to SDs temporarily. If the swap resulted in a decrease of the performance metric, the resulting solution is the new incumbent solution. Otherwise, the swap is reversed. After a predefined number of consecutive reversed swap operations, the algorithm terminates and returns the best solution found.

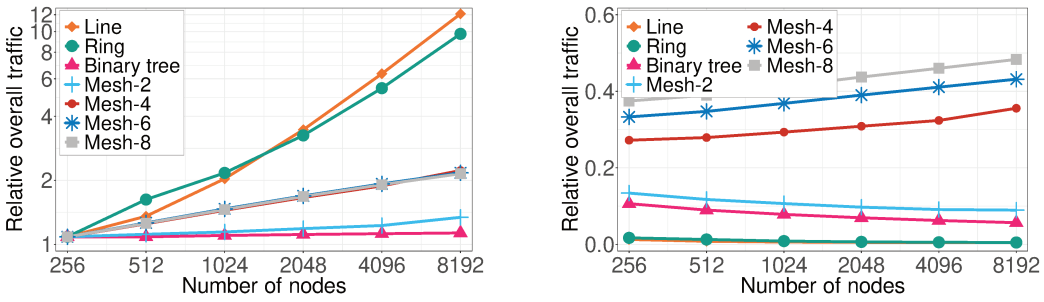
2.2.6.3 Results

The evaluation was twofold. First, we compared the heuristic to optimal and random SDs. Then, we leveraged the heuristic to evaluate the scalability of BIER as a mechanism independently of the used algorithm. All evaluations were performed for line, ring, tree, and random mesh-d topologies with constant node degree d, respectively.

Topology	n	s	Heuristic (%)	Random (%)
Mesh-2	64	2	100.3	132.6
		4	100.7	162.2
	128	2	100.5	133.7
		4	101.5	179.8
Mesh-4	64	2	100.3	115.2
Mesh-6	64	2	100.4	110.6
Mesh-8	64	2	100.3	107.1

Table 2.5: Overall traffic load for heuristic and random BIER clustering depending on network size n and number of subdomains s relative to optimal subdomains obtained by the ILP. Table adopted from [MSM23].

We compared the SDs computed by the heuristic with random SDs and optimal SDs obtained by the ILP model for small-scale network topologies. Table 2.5 compiles the resulting overall traffic loads relative to the traffic load of optimal SDs. The results indicate that the heuristic finds near-optimal solutions and that finding such solutions



(a) BIER relative to IPMC.

(b) BIER relative to IP unicast.

Figure 2.11: Overall traffic of BIER in various topologies. Figures from [MSM23].

is non-trivial as random SDs result in significantly more traffic. Unfortunately, the ILP was not able to compute solutions for network topologies with more than 128 nodes. However, we constructed optimal SDs for line, ring, and binary tree topologies with up to 8192 nodes manually and confirmed that the heuristic is effective in these cases, too. We conclude that the heuristic computes high-quality SDs that are suitable to evaluate the scalability of the BIER mechanism in general.

We compared BIER with traditional IPMC and IP unicast in various parameter studies. Figure 2.11a depicts the traffic that results from using BIER relative to the traffic of IPMC for varying numbers of nodes in a network topology. We observed that BIER is less efficient than IPMC. However, BIER results only in 2 times more traffic than IPMC in highly meshed and large topologies and even less in small topologies. Very sparse topologies as line and ring topologies benefit heavily from IPMC, but these topologies are unrealistic for real-world use cases of BIER. We observed similar results for another traffic model in which every node sends a multicast message only to a small subset of receivers. This can be considered as the price of less signaling overhead and state in core nodes compared to IPMC.

While BIER generates more traffic than IPMC, it is a significant improvement over IP unicast. Figure 2.11b shows the results of BIER relative to IP unicast. We observe that about 50% – 90% less traffic is transmitted when BIER is used. We conclude that BIER is a lightweight alternative to IPMC that is more efficient than unicast. More comparisons between BIER, IPMC, and IP unicast that support this conclusion can be found in Merlin et al. [MSM23] (cf. Appendix 1.4).

2.2.6.4 Conclusion & Discussion

In this work, we proposed an efficient algorithm to compute BIER subdomains in large networks. We compared the results of this algorithm to optimal solutions and concluded that the algorithm yields high quality solutions. Therefore, we were able to evaluate the scalability of the BIER mechanism in general. We demonstrated the costs imposed by BIER compared to IPMC and IP unicast in various parameter studies. These evaluations showed that BIER performs reasonably well in realistic topologies. IPMC is only in sparse topologies significantly more efficient than BIER. However, BIER is a big improvement over IP unicast in those topologies. Given the advantages of BIER such as reduced signaling overhead and less state in forwarding nodes, we conclude that BIER is a viable alternative for IPMC in large-scale networks.

A drawback of the presented work is the lack of evaluations with real-world topologies of large networks. Unfortunately, these topologies are not available due to restrictive information policies of large-scale content network providers. However, the topologies used in this work should be a good sample of sparse topologies (line and ring), hierarchical topologies (binary tree), and highly meshed topologies (mesh-d). Thus, we suspect that the results are applicable to real-world topologies as well.

2.2.7 Scalability of Segment-Encoded Explicit Trees (SEETs) for Efficient Stateless Multicast

This section summarizes the contributions of the research work from Lindner et al. [LSM24]. This publication is part of the core content of this thesis. We remark that the first two authors contributed equally to this publication. First, we introduce the problem description. Then, the encoding and the fragmentation algorithm are explained. Finally, the findings are summarized.

2.2.7.1 Problem Description

Stateless multicast protocols such as BIER reduce signaling overhead and state in core nodes compared to IPMC. However, BIER comes with several drawbacks inherited from IP. For instance, BIER employs the routing underlay of IP to determine packet paths. That means the path of a packet cannot be selected explicitly. Thus, load balancing to relieve bottleneck links is not possible. Some applications require reliable data

transmission even in case of failures. BIER has no mechanism to send a packet twice via disjoint paths to enhance reliability (1+1 protection). Additionally, the encoding of receivers with a bitstring is inefficient when the number of receivers is low. Most parts of the bitstring are irrelevant from the point of view of any forwarding node.

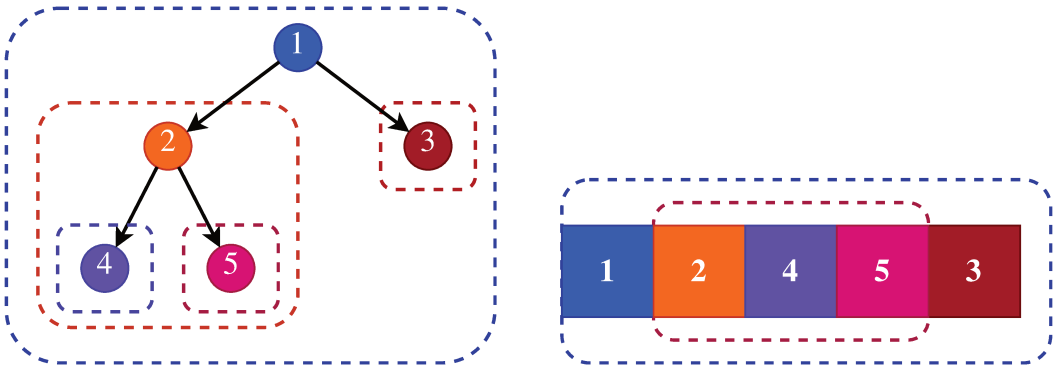
This work proposes Segment-Encoded Explicit Trees (SEET), a novel protocol for stateless multicast. SEET encodes the forwarding tree of a packet in the packet's header. Thus, packet paths are explicit and can be constructed to enable load balancing and 1+1 protection. Additionally, the header is decomposed in forwarding nodes and only the relevant parts are put in the header of replicated packets.

SEET allows to encode forwarding trees, i.e., explicit trees, in a packet's header. However, the size of a header is limited due to technical restrictions in forwarding nodes. Thus, the maximum size of a forwarding tree that can be encoded is limited. To cope with this problem, a set of receivers can be divided into subsets such that the forwarding tree of each subset can be encoded in a single packet header. This work proposes an algorithm for this purpose that minimizes the number of packet hops required to deliver a message to all receivers. We employ this algorithm to compare BIER and SEET in a quantitative study.

2.2.7.2 Concept

We introduce the encoding and the fragmentation algorithm.

2.2.7.2.1 Encoding We encode a forwarding tree by a list of nodes. Figures 2.12a–2.12b depict an example which will be used to explain the encoding. Node 1 is the source of the forwarding tree. Thus, it is the first node in the list. The nodes 2 and 3 are children of node 1. That means they are placed after node 1 in the list. However, they are not necessarily the direct successors of node 1. Instead, the encoding procedure is applied recursively to the subtrees rooted at nodes 2 and 3, respectively. The resulting node lists are appended after node 1. Thus, the encoding is a list of nodes in topological order starting with the root of the forwarding tree. The dashed rectangles in Figure 2.12b represent subtrees of the forwarding tree. These continuous sublists are denoted as recursive units. The size of the node list can be reduced by only including leaf nodes and replicating nodes, i.e., nodes with a fan-out of at least 2, in more complex examples.



(a) Forwarding tree. (b) Serialization of forwarding tree.

Figure 2.12: Encoding of a forwarding tree in a serialized header. The dashed rectangles indicate so-called recursive units.

Nodes are encoded by global identifiers, i.e., unique numbers for the network under consideration. Large networks result in identifiers with 12 – 14 bits length. Thus, each node requires at least 2 bytes of space in a SEET header. In contrast, BIER requires only a single bit to encode a receiving node. However, SEET only encodes nodes that are actually part of the forwarding tree while a BIER header contains one bit for every BFER.

Typical use cases of multicast protocols are distribution networks with hierarchical tree-like structure. Only the leaf nodes of these trees are receivers of messages. The penultimate hops of the leaf nodes in these networks have very high fan-out. Thus, we employ the idea of BIER bitstrings for an efficient encoding at the last hop. Instead of encoding nodes 4 and 5 by a separate entry in the list as in the example, the entry for node 2 contains a local bitstring when this optimization is applied. The bits of this bitstring represent neighbors of the penultimate hop, i.e., have local semantics. In this way it is possible to encode many receivers connected to the same penultimate hop efficiently.

2.2.7.2.2 Fragmentation Algorithm The input of the algorithm is a network topology, a source node, and a set of receivers. It fragments the set of receivers into subsets that can be addressed by a single packet. The algorithm leverages two observations about the encoding. First, nodes with a long common subpath should be put in the same packet. In this way, long distances in the network are tunneled with a single SEET header to the first replication node in the forwarding tree. Second, putting nodes with the same penultimate hop in the same packet does not impose additional costs as they are addressed with the same local bitstring.

The algorithm starts Dijkstra's algorithm at the source node. Every time a receiver node n is discovered, exactly one of the following conditions applies and the respective action is taken:

1. The current header is empty. A SEET header to n is introduced.
2. The current header does not contain a SEET header to a node on the path from the source to n. A SEET header to the penultimate hop of n is introduced.
3. The current header contains a SEET header to the penultimate hop p of n. The bit corresponding to n in the local bitstring of p is set to 1.
4. The current header contains a SEET header to a node n_2 with the same penultimate hop p as n. This SEET header is removed, a SEET header to p is introduced, and the bits corresponding to n and n_2 in the local bitstring of p are set to 1.
5. The current header contains a SEET header to some node that has a common subpath with n. A SEET header to the last possible replication node is inserted. A SEET header to n is inserted.

If the resulting header size exceeds the maximum allowed header length, the discovered node is not added to the current header. Instead, the current packet is finalized, and the node is added to a new empty header.

2.2.7.3 Results

First, we motivated the fragmentation algorithm by evaluating the header size of SEET packets. We showed that moderate numbers of receivers result in headers with more than 256 B length. Current P4-based switches cannot process longer headers due to technical restrictions. Thus, an algorithm to fragment a set of receivers into subsets that can be handled must be used.

Then, we employed the fragmentation algorithm to compare SEET with BIER and IPMC. We used the algorithm from Merlin et al. [MSM23] (cf. Appendix 1.4) to compute BIER subdomains. Figure 2.13 depicts the number of packet transmissions of SEET relative to BIER. We observed that SEET requires less packet transmissions than BIER except for very large multicast groups. This result seems counterintuitive as BIER can encode one receiver per bit in the bitstring. However, BIER subdomains are statically configured. In contrast, a SEET packet is constructed specifically for a given source node and set of receivers. IPMC requires less packet transmissions than SEET.

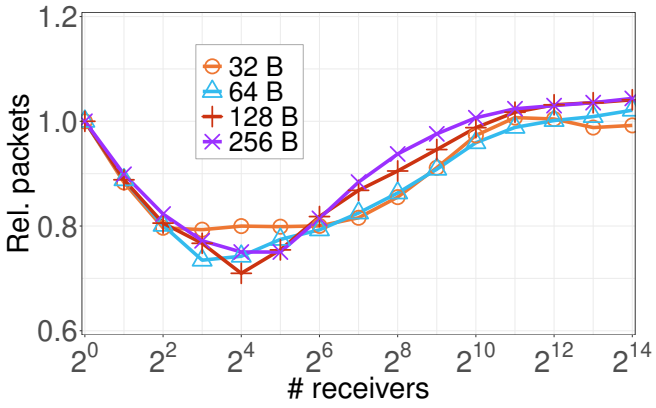


Figure 2.13: Number of individual packet transmissions of SEET in the network relative to BIER. Results for 32 B, 64 B, 128 B, and 256 B headers coincide. Adopted from [LSM24].

However, the reduction in packet transmissions is moderate (12%–70%), and IPMC is neither stateless nor does it support tree engineering.

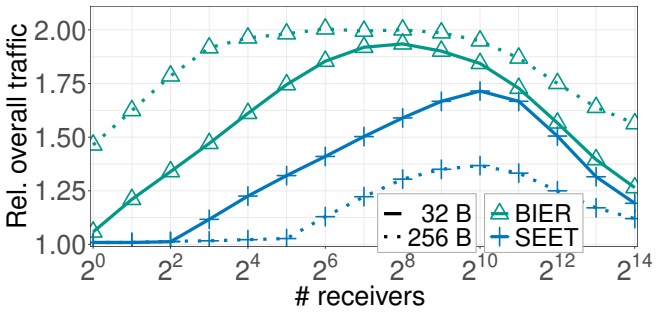


Figure 2.14: Overall traffic of BIER and SEET relative to IPMC, i.e., relative amount of data transmitted in the entire network. Results for 32 B and 256 B headers. Adopted from [LSM24].

Finally, we compared SEET and IPMC with respect to overall traffic transmitted in the network. Figure 2.14 depicts the overall traffic of BIER and SEET relative to IPMC. We showed that SEET results in less traffic than BIER for the considered topologies. This is because the size of the SEET header reduces along the path from the source node to the receivers as only the relevant parts are replicated. Additionally, many bits in the bitstring of a BIER packet are set to 0. Thus, the BIER header is not efficient for small and medium sized multicast groups.

2.2.7.4 Conclusion & Outlook

In this work, we proposed Segment-Encoded Explicit Trees (SEET), a novel protocol for stateless multicast. Additionally, we presented an algorithm to construct SEET packet headers. We employed the algorithm to compare SEET with BIER and IPMC. The evaluations showed that SEET is more efficient than BIER with respect to various metrics. Additionally, SEET supports tree engineering in contrast to BIER and IPMC. We conclude that SEET is a viable alternative for BIER.

Future works may investigate the usage of SEET in data centers and content provider networks. BIER is already successfully applied in such networks. We suspect that SEET will outperform BIER in these use cases, too.

There is a variant of BIER that supports tree-engineering denoted as BIER-TE. Like the fragmentation problem of SEET, BIER-TE also requires that a message with many receivers is fragmented into multiple packets. The fragmentation problem of BIER-TE is complex and there is no algorithm in the literature to solve it. Future works may propose such an algorithm and employ it to evaluate the scalability of BIER-TE as we have done for BIER [MSM23] and SEET [LSM24]. However, BIER-TE employs similar ideas as BIER. The authors suspect that the drawbacks of BIER compared to SEET will also apply for BIER-TE.

3 Additional Scientific Work

This chapter summarizes additional scientific contributions, which have been made during my doctoral studies besides the publications presented in Chapter 2.

3.1 Sustanability and Lectures for Future

The work presented in [SHM19] was preceded by my Master's thesis about the same topic. The Master's thesis was awarded with the sustainability award of the University of Tuebingen due to its contribution to the energy transition. Additionally, I participated in the Lectures for Future week in 2019 in the context of the lecture "Informatik der Systeme". The scope of the lecture was to increase the awareness for climate change among computer science students. Additionally, it featured a tutorial about power grids and power generation. The talk was repeated in the second instance of Lectures for Future and at the sustainability symposium of the computer science department in the same year. All following iterations of the lecture series "Informatik der Systeme" features this Lectures for Future contribution as regular chapter.

3.2 Research Proposals

I was involved in the creation of the DFG research proposal "Algorithms and Concepts for Time-Sensitive Networking (ACTSN)". Further, I was responsible for selected parts of multiple technical reports and deliverables for the Collaborative Project KITOS (support code 16KIS1161).

3.3 Thesis Supervision

I supervised two Master theses and four Bachelor theses during my doctoral studies. Topics included the design and implementation of forecasting and optimization algo-

rithms in the context of power systems, BIER, and TSN.

B.Sc. ”Vergleich von Prognosemethoden für unterschiedliche Energiebedarfe in Einfamilienhäusern“

B.Sc. ”Entwicklung eines Tools zur Korrektur von Programmieraufgaben in MIPS-Assembler“

B.Sc. ”Entwicklung und Implementierung eines Frameworks zur Optimierung von Netzwerk-Segmentierung für BIER und BIER-TE mit Hilfe von Evolutionären Algorithmen“

B.Sc. ”Design und Vergleich von Clustering-Algorithmen zur Verbesserung der Skalierbarkeit von BIER-basiertem Multicast in Kommunikationsnetzen“

M.Sc. ”Design eines ILP zur Lösung von Scheduling-Problemen in TSN und Untersuchung ihrer Lösbarkeit in Abhängigkeit von Modellparametern“

M.Sc. ”Der SSteP-KiZ Prototyp: Ein Werkzeug zur multimodalen Echtzeit-Interaktion für die Tele-Psychotherapie von Kindern im häuslichen Umfeld“

Two of the supervised Master theses have laid the foundation for the start of independent Ph.D. topics of new co-workers, Manuel Eppler and Jonas Primbs.

3.4 Miscellaneous

During my doctoral studies, I supervised the lectures ”Informatik der Systeme” (3 times) and ”Network Security” (3 times). I gave lectures on cryptography, Bitcoin, Blockchain, and Tor in our course ”Network Security”.

From 2019-2024, I was a reviewer for the following international journals, magazines, conferences, and workshops:

- IEEE Access (2023, 2024)
- IEEE Transactions on Network and Service Management (TNSM 2023)
- IEEE Transactions on Industrial Informatics (2024)
- International Conference on Networked Systems (NetSys 2019)
- International Conference on the Design of Reliable Communication Networks (DRCN 2019, 2020, 2021)
- International Teletraffic Congress (ITC 2021, 2022)

Bibliography

[80291] Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges. *IEEE Std 802.1D-1990*, pages 1–176, 1991.

[80210] IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams. *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pages C1–72, 2010.

[80216] IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic. *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pages 1–57, 2016.

[80217a] IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing. *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)*, pages 1–65, 2017.

[80217b] IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability. *IEEE Std 802.1CB-2017*, pages 1–102, 2017.

[80218] IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks. *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pages 1–1993, 2018.

[BSN+14] Marc Boyer, Luca Santinelli, Nicolas Navet, Jörn Migge, and Marc Fumey. Integrating End-System Frame Scheduling for More Accurate AFDX Timing Analysis. In *Embedded Real Time Software and Systems (ERTS)*, 2014.

[COCS16] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks. In *International Conference on Real-Time Networks and Systems (RTNS)*, pages 183–192, 2016.

[DN16] Frank Dürr and Naresh Ganesh Nayak. No-wait Packet Scheduling for IEEE Time-Sensitive Networks (TSN). In *International Conference on Real-Time Networks and Systems (RTNS)*, 2016.

[DPSW00] Ralf Diekmann, Robert Preis, Frank Schlimbach, and Chris Walshaw. Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM. *Parallel Computing*, 26(12):1555–1581, 2000.

[dSSN19] Aellison Cassimiro T dos Santos, Ben Schneider, and Vivek Nigam. TSNSCHED: Automated Schedule Generation for Time Sensitive Networking. In *Formal Methods in Computer Aided Design (FMCAD)*, pages 69–77, 2019.

[FDR18] Jonathan Falk, Frank Dürr, and Kurt Rothermel. Exploring Practical Limitations of Joint Routing and Scheduling for TSN with ILP. In *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 136–146, 2018.

[FDR20] Jonathan Falk, Frank Dürr, and Kurt Rothermel. Time-Triggered Traffic Planning for Data Networks with Conflict Graphs. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 124–136, 2020.

[GZRP18] Voica Gavriluţ, Luxi Zhao, Michael L Raagaard, and Paul Pop. AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN. *IEEE Access*, 6:75229–75243, 2018.

[HGF+20] David Hellmanns, Alexander Glavackij, Jonthan Falk, René Hummen, Stephan Kehrer, and Frank Dürr. Scaling TSN Scheduling for Factory Automation Networks. In *IEEE International Conference on Factory Communication Systems (WFCS)*, pages 1–8, 2020.

[HHS+19] Florian Heimgaertner, Sascha Heider, Thomas Stüber, Daniel Merling, and Michael Menth. Load Profile Negotiation for Compliance with Power Limits in Day-Ahead Planning. In *International ETG-Congress 2019;*

ETG Symposium, pages 1–6, 2019. ©2019 VDE Verlag GmbH. Reprinted with permission.

[Ind18] Industrial Internet Consortium. Time Sensitive Networks for Flexible Manufacturing Testbed - Description of Converged Traffic Types, 2018. [Online; accessed 21-September-2023].

[JXG+20] Xi Jin, Changqing Xia, Nan Guan, Chi Xu, Dong Li, Yue Yin, and Peng Zeng. Real-Time Scheduling of Massive Data in Time Sensitive Networks With a Limited Number of Schedule Entries. *IEEE Access*, 8:6751–6767, 2020.

[KZH15] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. Real World Automotive Benchmarks for Free. In *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[LSM24] Steffen Lindner, Thomas Stüber, and Michael Menth. Scalability of Segment-Encoded Explicit Trees (SEETs) for Efficient Stateless Multicast, 2024. A preliminary version can be found in the Appendix.

[MMWE18] Daniel Merling, Michael Menth, Nils Warnke, and Toerless. Eckert. An Overview of Bit Index Explicit Replication (BIER). In *IETF Journal*, March 2018.

[MSM23] Daniel Merling, Thomas Stüber, and Michael Menth. Efficiency of BIER Multicast in Large Networks. *IEEE Transactions on Network and Service Management (TNSM)*, 20(4):4013–4027, 2023. ©2022 IEEE. Reprinted with permission. `https://doi.org/10.1109/TNSM.2023.3262294`.

[OCS18] Ramon Serna Oliver, Silviu S Craciunas, and Wilfried Steiner. IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 13–24, 2018.

[PO18] Maryam Pahlevan and Roman Obermaisser. Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks. In *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 337–344, 2018.

[PSRNH15] Francisco Pozo, Wilfried Steiner, Guillermo Rodriguez-Navas, and Hans Hansson. A decomposition approach for SMT-based schedule synthesis for time-triggered networks. In *IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, 2015.

[PTO19] Maryam Pahlevan, Nadra Tabassam, and Roman Obermaisser. Heuristic List Scheduler for Time Triggered Traffic in Time Sensitive Networks. *ACM SIGBED Review*, 16(1):15–20, 2019.

[RN10] Stuart J Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. London, 2010.

[SEOM24] Thomas Stüber, Manuel Eppler, Lukas Osswald, and Michael Menth. Performance Comparison of Offline Scheduling Algorithms for the Time-Aware Shaper (TAS). *IEEE Transactions on Industrial Informatics (TII)*, pages 1–13, 2024. Early access; ©2024 IEEE. Reprinted with permission. `https://doi.org/10.1109/TII.2024.3385503`.

[SHM19] Thomas Stüber, Florian Heimgaertner, and Michael Menth. Day-Ahead Optimization of Production Schedules for Saving Electrical Energy Costs. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems (e-Energy '19)*, page 192–203, 2019.

[SHTM21] Thomas Stüber, Ricarda Hogl, Bernd Thomas, and Michael Menth. Comparison of Forecasting Methods for Energy Demands in Single Family Homes. In *ETG Congress 2021*, pages 1–5, 2021. ©2021 VDE Verlag GmbH. Reprinted with permission.

[SOLM23] Thomas Stüber, Lukas Osswald, Steffen Lindner, and Michael Menth. A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN). *IEEE Access*, 11:61192–61233, 2023. Reprinted with permission. `https://doi.org/10.1109/ACCESS.2023.3286370`.

[SOM24] Thomas Stüber, Lukas Osswald, and Michael Menth. Efficient Robust Schedules (ERS) for Time-Sensitive Networking, 2024. Accepted for publication in a future issue of the IEEE Open Journal of the Communications Society journal.

[STM23] Thomas Stüber, Bernd Thomas, and Michael Menth. Minimizing Grid Electricity Consumption and On-/Off-Cyles for Heat Pumps in Single-Family Homes with PV Panels, 2023. Submission to the Applied Thermal Engineering journal on 2023-10-02.

[STP+20] Eike Schweissguth, Dirk Timmermann, Helge Parzyjegla, Peter Danielis, and Gero Mühl. ILP-Based Routing and Scheduling of Multicast Real-time Traffic in Time-Sensitive Networks. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–11, 2020.

[WRD+17] IJsbrand Wijnands, Eric C. Rosen, Andrew Dolganow, Tony Przygienda, and Sam Aldrin. RFC8279: Multicast Using Bit Index Explicit Replication (BIER). Request for comments, Internet Engineering Task Force, November 2017. `https://www.rfc-editor.org/info/rfc8279`.

Personal Contribution

Accepted Manuscripts (Core Content)

1. **Day-Ahead Optimization of Production Schedules for Saving Electrical Energy Costs** [SHM19]

Scope of the joint work	This research work was done in the context of the research project „Entwicklung einer verteilten Regelarchitektur zur Einbindung indirekt steuerbarer Verbraucher/Erzeuger in virtuelle Kraftwerke“ funded by the German Federal Ministry for Economic Affairs and Energy under grant no. 16KN039521. The scope of this work was developing and implementing an algorithm to minimize energy costs for large scale production processes when energy is purchased at the day-ahead market.
Names of collaborators and their shares	Florian Heimgärtner: Editorial assistance for writing the publication. Michael Menth: Scientific supervision and editorial assistance on the publication.
Importance of own contributions to the joint work	Main developer of the algorithm. Responsible for the implementation and evaluation. Main author of the publication. Presented the results at the conference.

2. **A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN)** [SOLM23]

Scope of the joint work	This research work was done in the context of the research project „Künstliche Intelligenz zur dynamischen Optimierung des Netzwerkmanagements (KITOS)“ of the German Federal Ministry of Education and Research (BMBF) under grant no. 16KIS1161. The scope of this work was to review all published works about scheduling with the TAS and to compile the contributions of these works.
Names of collaborators and their shares	Lukas Osswald: Editorial assistance and co-author of the publication. Steffen Lindner: Editorial assistance and co-author of the publication. Michael Menth: Scientific supervision and editorial assistance on the publication.
Importance of own contributions to the joint work	Main author of the survey. Conducted all literature work including reading, classifying, and summarizing the papers.

3. **Performance Comparison of Offline Scheduling Algorithms for the Time-Aware Shaper (TAS)** [SEOM24]

Scope of the joint work	This research work was done in the context of the research project „Künstliche Intelligenz zur dynamischen Optimierung des Netzwerkmanagements (KITOS)“ of the German Federal Ministry of Education and Research (BMBF) under grant no. 16KIS1161. The scope of this work was twofold. First, a set of problem instances for the comparison of different scheduling algorithms and future works should be constructed. Second, various scheduling algorithms from the literature should be compared in a systematic and reproducible way.

Names of collaborators and their shares	Manuel Eppler: Discussion, feedback, and co-author of the publication. Manuel Eppler: Discussion, feedback, and co-author of the publication. Michael Menth: Scientific supervision and editorial assistance on the publication.
Importance of own contributions to the joint work	Main author of the publication. Construction of the problem instances. Implementation of the compared algorithms and the evaluations.

4. **Efficiency of BIER Multicast in Large Networks** [MSM23]

Scope of the joint work	This research work was done in the context of the research project „Future Internet Routing (FIR)“ funded by the Deutsche Forschungsgemeinschaft (DFG) under grant no. ME2727/1-2. The scope of this work was developing and implementing an algorithm to compute optimal BIER subdomains in large networks.
Names of collaborators and their shares	Daniel Merling: Joint main author of the publication. Editorial assistance for writing the publication and discussions about evaluation design. Michael Menth: Scientific supervision and editorial assistance on the publication.
Importance of own contributions to the joint work	Main developer of the algorithm. Responsible for the implementation and evaluation. Joint main author of the publication with Daniel Merling.

Submitted Manuscripts (Core Content)

5. **Minimizing Grid Electricity Consumption and On-/Off-Cyles for Heat Pumps in Single-Family Homes with PV Panels** [STM23]

Scope of the joint work	This research work was done in an informal cooperation with the Reutlingen University. The scope of this work was to develop a scheduling algorithm which minimizes multiple objectives simultaneously in a rolling horizon approach over the course of a year.
Names of collaborators and their shares	Bernd Thomas: Related work, feedback, data set acquisition, and scientific supervision. Michael Menth: Scientific supervision and editorial assistance on the publication.
Importance of own contributions to the joint work	Main author of the paper. Main developer of the algorithm. Responsible for the implementation and evaluation.

6. **Efficient Robust Schedules (ERS) for Time-Sensitive Networking** [SOM24]

Scope of the joint work	This research work was done in the context of the research project „Künstliche Intelligenz zur dynamischen Optimierung des Netzwerkmanagements (KITOS)“ of the German Federal Ministry of Education and Research (BMBF) under grant no. 16KIS1161. The scope of this work was to develop a scheduling algorithm which produces schedules that are robust against common sources of non-determinism such as time synchronization errors and processing jitter.
Names of collaborators and their shares	Lukas Osswald: Editorial assistance and co-author of the publication. Michael Menth: Scientific supervision and editorial assistance on the publication.
Importance of own contributions to the joint work	Main author of the paper. Main developer of the algorithm. Responsible for the implementation and evaluation.

7. **Scalability of Segment-Encoded Explicit Trees (SEETs) for Efficient Stateless Multicast** [LSM24]

Scope of the joint work	The scope of this work was to develop a novel stateless multicast protocol that allows tree engineering. Additionally, an algorithm to construct packet headers for the protocol should be developed. This algorithm was employed to conduct a quantitative comparison study of the new protocol and BIER.
Names of collaborators and their shares	Steffen Lindner: Responsible for multiple sections of the paper and the SDN implementation. Planning of evaluations and discussion of results. Toerless Eckert: Scientific supervision and discussions about the protocol. Michael Menth: Scientific supervision and editorial assistance on the publication.
Importance of own contributions to the joint work	Second main author of the paper. Main developer of the algorithm. Responsible for the implementation and evaluation.

Accepted Manuscripts (Additional Content)

8. **Comparison of Forecasting Methods for Energy Demands in Single Family Homes** [SHTM21]

Scope of the joint work	This research work was done in the context of the research project „Entwicklung einer verteilten Regelarchitektur zur Einbindung indirekt steuerbarer Verbraucher/Erzeuger in virtuelle Kraftwerke“ funded by the German Federal Ministry for Economic Affairs and Energy under grant no. 16KN039521. The scope of this work was the evaluation of different forecasting strategies as a preliminary work for [STM23].

Names of collaborators and their shares	Ricarda Hogl: Preliminary works and discussions. Bernd Thomas: Feedback, data set acquisition, and scientific supervision. Michael Menth: Scientific supervision and editorial assistance on the publication.
Importance of own contributions to the joint work	Main author of the publication. Responsible for implementation and evaluation. Presented the results at the workshop.

9. **Load Profile Negotiation for Compliance with Power Limits in Day-Ahead Planning**[HHS+19]

Scope of the joint work	This research work was done in the context of the research project „Entwicklung einer verteilten Regelarchitektur zur Einbindung indirekt steuerbarer Verbraucher/Erzeuger in virtuelle Kraftwerke“ funded by the German Federal Ministry for Economic Affairs and Energy under grant no. 16KN039521. The scope of this work was the development of a mechanism to negotiate load profiles between multiple consumers and an aggregator in compliance with power limits.
Names of collaborators and their shares	Florian Heimgärtner: Main author of the publication, taking on most of the write-up. Sascha Heider: Preliminary work and study of the problem. Daniel Merling: Discussion and feedback. Michael Menth: Scientific supervision and editorial assistance on the publication.

Importance of own contributions to the joint work	Editorial assistance on the publication. Development of the ILP models and validation of the results obtained in a previous bachelor's thesis on the topic.

Publications

1 Accepted Manuscripts (Core Content)

1.1 Day-Ahead Optimization of Production Schedules for Saving Electrical Energy Costs

Day-Ahead Optimization of Production Schedules for Saving Electrical Energy Costs

Thomas Stueber
Chair of Communication Networks, University of Tuebingen
Tuebingen, Germany
thomas.stueber@uni-tuebingen.de

Florian Heimgaertner
Chair of Communication Networks, University of Tuebingen
Tuebingen, Germany
florian.heimgaertner@uni-tuebingen.de

Michael Menth
Chair of Communication Networks, University of Tuebingen
Tuebingen, Germany
menth@uni-tuebingen.de

ABSTRACT

The integration of weather-dependent renewable energy sources leads to an increased volatility of electrical energy supply. As a result, considerable intra-day price spreads can be observed at the spot markets for electrical energy. To benefit from variable energy prices, enterprises can use price forecasts for cost-optimized load scheduling. Thereby energy costs can be reduced by shifting energy-intensive processes to times with lower energy prices.

In this work, we propose a method to model an industrial unit including devices, storage units, dependencies, restrictions, and production targets as a mixed integer linear program (MILP). Along with a time series of energy prices, the MILP is used to compute optimal run times for the devices while complying with the specified restrictions.

We use the model of a cement plant as an example. We show potential savings compared to default schedules over individual day, weeks, or over the year 2018. We propose optimization with look-ahead, point out its benefits compared to optimization without look-ahead, and show the influence of storages sizes and price variance on the savings potential.

CCS CONCEPTS

• **Theory of computation** → **Linear programming**; • **Hardware** → **Smart grid**; • **Applied computing** → *Industry and manufacturing*.

ACM Reference Format:
Thomas Stueber, Florian Heimgaertner, and Michael Menth. 2019. Day-Ahead Optimization of Production Schedules for Saving Electrical Energy Costs. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems (e-Energy '19), June 25–28, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3307772.3328302

1 INTRODUCTION

The large-scale integration of renewable energy sources leads to new challenges for electrical power grids and the energy market. Especially the roll-out of weather-dependent energy sources like wind turbines or photovoltaic systems leads to an increased volatility of electrical energy supply. As a result, considerable differences in energy prices can be observed at the spot markets for electrical energy within a day.

In industrial production processes, some devices can be operated at different production rates. Additionally, storage units can decouple the run times of subsequent devices within a production chain. Storage and variable production rates constitute a certain flexibility, i.e., an enterprise can reach the same production targets with the same set of devices while using different schedules. With high-quality price forecasts for the day-ahead markets, industrial enterprises can leverage flexibilities in their production processes to benefit from the variability of energy prices using cost-optimized load scheduling. Based on the forecasts, energy costs can be reduced by shifting energy-intensive processes to times with lower energy prices [4].

In this work, we propose a method for the computation of production schedules that minimize energy costs. The main contribution of our work is a comprehensive framework for modeling an industrial plant including devices, storage units, dependencies, restrictions, and production targets as a mixed integer linear program (MILP). With a time series of energy prices, the MILP computes optimized run times for the devices with the given production rates, storage parameters and restrictions.

Cement production is a prominent example for energy intensive industry, accounting for approximately 12–15% of the total industrial energy consumption [8]. As shown in Section 2.2 cement production is also widely used as a reference use case for scheduling of energy intensive processes. For our study we use the model of a cement plant described by Bazan et. al. [1] as an example to show potential savings compared to default schedules. We quantify the savings for individual days, weeks, and for the year 2018. As another contribution, we propose optimization with look-ahead for this problem and demonstrate its benefits compared to optimization without look-ahead. In addition, we show the influence of storage sizes and price variance on the savings potential. Finally, we report the computation time of our optimization approach.

This paper is structured as follows. Section 2 discusses related work. In Section 3 we describe the proposed approach for modeling of industrial production processes as MILP. Section 4 introduces the evaluation scenario and discusses optimization results. Section 5 draws conclusions and gives an outlook on further work.

e-Energy '19, June 25–28, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6671-7/19/06.
https://doi.org/10.1145/3307772.3328302

2 RELATED WORK

In this section we give an overview about related work. We first discuss general approaches for scheduling in the area of continuous production. Then, we discuss work in the area of energy-aware scheduling, with a focus on scheduling approaches specifically addressing the use case of cement production.

2.1 Continuous Scheduling

Different scheduling strategies for continous production processes are considered in literature. They can be classified by algorithmic techniques, time model and objective. Some of the approaches also support semi-continous or batch processes.

One of the first such models for short-term scheduling of the production of fast moving consumer goods was presented by Ierapetritou and Floudas [5]. The optimization is based on MILP. This model was extended for storages and used to examine properties of the optimization with different storage limitations by Mendez and Cerda [9] and later by Shaik and Floudas [13]. Neumann and Schwindt [11] propose a branch-and-bound algorithm for models with continous and semi-continous processes.

However, these models were designed for optimizing makespans and cost of production processes and do not consider energy cost or usage.

2.2 Energy-Aware Scheduling

Castro et. al. [2] explore different scheduling approaches for continuous production. They use both discrete-time and continuous-time models. Energy consumption is considered in the optimization, but variable energy prices are not used. Shrouf et. al. [14] present an energy-aware scheduling mechanism using linear programming (LP) and a discrete-time model. The optimization objective is reduced energy costs with day-ahead energy prices. However, the scheduling only considers a single device.

Kondili et. al. [7] present an optimization of schedules for whole continous production plants with varying energy prices using MILP. A cement plant is used as an example for real world applications of such models. Based on [2], Mitra et. al. [10] present a model for a cement plant which also addresses variable energy prices. More recent developments in energy market models like day-ahead markets give new objecives in scheduling such processes. Bazan et. al. [1] present a hybrid simulation approach for scheduling of energy demand in a cement plant with a wind turbine and a battery storage. They optimize energy costs using LP and a discrete time model.

Gahm et. al. [3] provide a wide overview of the field of energy-aware scheduling in manufacturing companies.

In contrast to the works mentioned above, this work focuses on the optimization potential which arises from variable energy prices in day-ahead markets. We give a modelling approach for optimizing production schedules of complex production processes instead of single machines. The approach is used to gain first insights for the potential of saving energy costs by taking advantage of flexibilities in production combined with variability of energy prices.

3 MODELING FRAMEWORK

Industrial production processes are defined by devices, storage units, energy and material flows, and technical or organizational constraints. In this paper, we propose a comprehensive framework to model production processes as a MILP that can be used for process scheduling with minimized energy costs. In this section, we first describe the main components of our abstract model for an industrial plant. Then, we explain how they are represented in the mathematical model.

3.1 Model Components

We consider production processes in industrial facilities with continuous production and develop a simple abstract model which is powerful enough to describe many relevant degrees of freedom and restrictions for scheduling. The model consists of a set of devices $\mathcal{D}$, a set of storages $\mathcal{S}$, and a set of fixed consumers $\mathcal{F}$ that are connected by material and energy flows like a directed graph. Additional constraints for scheduling are specified by a set of global restrictions $\mathcal{G}$. In the following, we describe the model components in detail.

3.1.1 Devices $\mathcal{D}$. Devices consume and produce goods and power. The input and output volumes of goods and power depend on the operation mode of the device. The run times and the operation modes constitute the degrees of freedom of our scheduling problem. Various limitations can restrict the set of possible schedules of a device, e.g., prohibited or mandatory run times, preparation and waiting times before and after runs, maximum number of runs, or minimum and maximum production within a run or during the optimization interval.

3.1.2 Storages $\mathcal{S}$. Storages store goods or energy before and after devices in the production process. They cause time dependencies in the model, increasing the complexity of the scheduling problem. However, sufficiently large storages between devices decouple their operation in time and generate scheduling flexibility. Like devices, storages are subject to a set of restrictions, e.g., minimum and maximum level, optional production targets at different points in time, and starting levels.

3.1.3 Fixed Consumers $\mathcal{F}$. Fixed consumers are unscheduled parts of the production process. They can describe constant energy demands and supply of goods needed for production. They can be active only at a specific point in time or during longer time intervals. Fixed energy demands increase the energy costs only by a constant addend, but can be important for compliance with global restrictions.

3.1.4 Global Restrictions $\mathcal{G}$. Global restrictions are constraints that cannot be specified as a property of a single device, e.g., restrictions that apply to multiple devices at the same time. The model currently supports mutual exclusion of arbitrary subsets of devices and global energy and power limits.

3.2 The Mathematical Model

The general problem of computing a schedule for a given production process and a time series of energy prices with minimum energy cost is NP-hard. We provide a proof for that in Appendix A. This

Table 1: Global parameters.

Parameter	Definition
$\mathcal{T}, l$	Index set of all time slots (numbers 1 to $\|T\|$) and length of a time slot
$\mathcal{D}, \mathcal{S}, \mathcal{F}, \mathcal{G}$	Sets of devices, storages, fixed consumers, global restrictions
c_t	Energy price in time slot t

Table 2: Variables.

Variable	Type	Definition
$x^d_{t,m}$	binary	Does device d run in time slot t with mode m?
s^d_t	binary	Does a run of device d start at the beginning of time slot t?
e^d_t	binary	Does a run of device d end at the end of time slot t?
$k^d_{t,s}$	real	Cumulative production of the current run of device d after time slot t for output storage s.
f^s_t	real	Fill level of storage s at the end of time slot t.
$r^d_{t,m}$	real	Production rate of device d in continuous mode m during time slot t.

property makes the problem very unlikely to be solved by algorithms with polynomial runtime. Therefore, we use MILP to solve the problem although MILP solving algorithms have exponential runtime in general.

In the following, we present global parameters, variables, restrictions, and the objective function of our MILP and discuss its design.

3.2.1 Global Parameters. The MILP computes an optimized schedule for an optimization interval. Like in other MILP models for similar problems [1, 2, 14], the optimization interval is divided into a set of time slots $\mathcal{T}$ and all time slots $t \in \mathcal{T}$ have the same duration l. However, the latter can be easily relaxed. Energy price forecasts take a fixed value during time slots and are given by c_t. Table 1 summarizes all global parameters of the MILP.

3.2.2 Variables. Every device d is modeled with three binary variables and one continuous variable per time slot t. The binary variable $x^d_{t,m}$ indicates whether a device d runs during time slot t. Moreover, the continuous variable $r^d_{t,m}$ indicates the rate of device d when it works in operation mode m in time slot t.

The binary variable s^d_t indicates whether a run of a device d begins at the start of the time slot t. The binary variable e^d_t indicates whether a run of a device d ends at the end of time slot t. The variable $r^d_{t,m}$ indicates the operation mode m of a device d in time slot t. The continuous variable $k^d_{t,s}$ captures the cumulative production of device d for its connected storage s from the beginning of its run until the end of the current time slot t. The continuous variable f^s_t captures the fill state of storage s at the end of time slot t. Table 2 gives a summary of the used variables.

3.2.3 Restrictions. We first discuss implicit restrictions of our model and then explicit restrictions for devices, storages, fixed consumers, and global restrictions, which were all mentioned in Section 3.1 that are enforced by additional inequalities.

Implicit Restrictions. An essential restriction of our model is that devices run in the same operation mode during a time slot. This limitation facilitates modeling of storages. Their fill states at the end of a time slot can be computed from the level at the beginning of the same time slot and the sum of all devices which charge or discharge the storage during that slot. Furthermore, it facilitates a simple restriction for minimum and maximum fill states. As devices run for entire time slots with constant rates, storage levels are increased or decreased linearly during a time slot. Therefore, ensuring minimum and maximum fill levels at the ends of all time slots is sufficient to comply with restrictions also within time slots.

Technical Restrictions. Some inequalities are needed to enforce the semantics of the variables mentioned in Section 3.2.2. They are presented in Appendix B as they are of technical nature and are not used to model features.

Device Restrictions. Devices may be connected to several storages from which they receive input or to which they deliver output. We denote the set of all storages of a device d, to which it delivers output, by O^d. The restrictions for a device d require parameters given in Table 3 and are expressed as follows:

$$\sum_{t \in T} s^d_t \leq n^d_{start} \tag{1}$$

$$\forall o \in O^d : w^d_{o,min} \leq \sum_{t \in \mathcal{T}} l \cdot \left(\sum_{m \in \mathcal{M}_s} x^d_{t,m} \cdot m(o) + \sum_{m \in \mathcal{M}_c} r^d_{t,m} \cdot \mathrm{eff}^d_{o,m} \right) \leq w^d_{o,max} \tag{2}$$

$$\forall s \in O^d \forall t \in \mathcal{T} : k^d_{t,s} \leq v^d_{s,max} \tag{3}$$

$$\forall s \in O^d \forall t \in \mathcal{T} : |\mathcal{T}| \cdot l \cdot Max^d \cdot (e^d_t - 1) \leq k^d_{t,s} - v^d_{s,min} \tag{4}$$

$$\forall t \in \mathcal{T}_{must} : \sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{t,m} = 1 \tag{5}$$

$$\forall t \in \mathcal{T}_{forb} : \sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{t,m} = 0 \tag{6}$$

Inequation (1) guarantees that the number of starts, which is also the number of runs, cannot exceed n^d_{start}. While Inequation (2) holds, the global production for every output fullfills the demanded minimum and maximum production. Inequation (3) ensures that the cumulative production of a run stays below the maximum allowed production. Inequation (4) implies that the cumulative production of the run is larger than the minimum production per run if a run finishes in the respective time slot. Mandatory and prohibited times can be encoded with (5) and (6) by forcing the respective run variables to be 0 or 1.

Table 3: Device parameters.

Parameter	Definition
n^d_{start}, c^d_{start}	Maximum number of runs and startup cost of device d
$w^d_{s,max}, w^d_{s,min}$	Maximum/minimum global production for output storage s of device d
$v^d_{s,max}, v^d_{s,min}$	Maximum/minimum production per run for output storage s of device d
$\mathcal{T}^d_{must}, \mathcal{T}^d_{forb}$	Set of time slots in which device d must/must not run
t^d_{lead}, t^d_{over}	Number of time slots device d has to wait before/after a run
$\mathcal{M}^d_s$	Set of semi-continuous modes of device d. A semi-continuous mode is a mapping of inputs/outputs of the device to production rates.
$\mathcal{M}^d_c$	Set of continuous modes of device d. A continuous mode m is a 2-tuple with m_{min}, m_{max} being the minimal/maximal production rate in this mode.
$\text{eff}^d_{s,m}$	The factor of production rate of continuous mode m to procution input/output of storage s
O^d	Set of output storages of device d
Max^d	A number which is bigger than the production to all outputs if the machines runs the whole planning horizon
P^d_m	Power input of device d in semi-continuous mode m or energy efficiency for continuous mode m

Table 4: Storage parameters.

Parameter	Definition
f^s_{min}, f^s_{max}	Minimum/maximum fill level of storage s
f^s_0	Initial fill level of storage s
f^s_{prod}	Target fill level of storage s at the end of the planning horizon
$\mathcal{I}^s, O^s$	Set of charging/discharging devices of storage s

Lead time before the run of a device can be modeled by the inequality:

$$s^d_t \leq 1 - \sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{t',m} \tag{7}$$

which must hold for all $t \in \mathcal{T}$ and all t' with $t - t^d_{lead} \leq t' \leq t - 1$. Overrun after a run can be modeled analogously.

Restrictions for Storages. Storages are charged and discharged by devices or fixed consumers. The set of all devices charging a storage is its set of input devices $\mathcal{I}^s$ and the set of all devices discharging it is its set of output devices O^s. Fixed consumers may be used to model constant charging or discharging of a storage.

Table 5: Parameters for fixed consumers.

Parameter	Definition
$\mathcal{T}^F_{con}$	Set of time slots, in which fixed consumer F is active
R^F_S	Consumption rate of fixed consumer F from storage S
P^F	Power input of fixed consumer F

Table 6: Parameters for global restrictions.

Parameter	Definition
$\mathcal{T}^R_{lim}$	Set of time slots, in which global restriction R must hold
$E^R_{max}, P_{max}(R)$	Maximum amount of energy or peak power for global restriction R
$\mathcal{D}^R$	Set of devices, of which at most can run at the same time by global restriction R

The new fill level of a storage s at the end of a time slot t is given by

$$\begin{aligned} \forall t \in \mathcal{T} : f^s_t &= f^s_{t-1} \\ &+ \sum_{i \in \mathcal{I}^s} \left(\sum_{m \in \mathcal{M}^i_s} l \cdot m(s) \cdot x^i_{t,m} + \sum_{m \in \mathcal{M}^i_c} l \cdot r^i_{t,m} \cdot \text{eff}^i_{s,m} \right) \\ &- \sum_{o \in O^s} \left(\sum_{m \in \mathcal{M}^o_s} l \cdot m(s) \cdot x^o_{t,m} + \sum_{m \in \mathcal{M}^o_c} l \cdot r^o_{t,m} \cdot \text{eff}^o_{s,m} \right) \\ &- \sum_{F \in \mathcal{F} : t \in \mathcal{T}^F_{con}} R^F_s . \end{aligned} \tag{8}$$

The equation considers the charging of a storage by all its input devices, the discharging by all its output devices, and the charging or discharging by all fixed consumers that are active in time slot t. If an active fixed consumer F does not charge or discharge a storage s, this is expressed by $R^F_s = 0$.

Storage level and production targets can be expressed by the following inequality:

$$f^s_{|T|} \geq f^s_{prod} \tag{9}$$

$$\forall t \in \mathcal{T} : f^s_{min} \leq f_t \leq f^s_{max} \tag{10}$$

Global Restrictions. A global restriction R enforcing a maximum of used energy (11), maximum peak power (12), or mutual exclusion of at most k devices of a set of devices (13) is implemented,

depending on the respective type, with the following inequations:

$$\sum_{t \in \mathcal{T}^R_{lim}} \sum_{d \in \mathcal{D}} \left(\sum_{m \in \mathcal{M}^d_s} x^d_{t,m} \cdot P^d_m \cdot l + \sum_{m \in \mathcal{M}^d_r} r^d_{t,m} \cdot P^d_m \cdot l \right) \leq E^R_{max} \tag{11}$$

$$t \in \mathcal{T}^R_{lim} : \sum_{d \in \mathcal{D}} \left(\sum_{m \in \mathcal{M}^d_s} x^d_{t,m} \cdot P^d_m + \sum_{m \in \mathcal{M}^d_r} r^d_{t,m} \cdot P^d_m \right) \leq P^R_{max} \tag{12}$$

$$t \in \mathcal{T}^R_{lim} : \sum_{d \in \mathcal{D}^R} \sum_{m \in \mathcal{M}^d_s \cup \mathcal{M}^d_r} x^d_{t,m} \leq k \tag{13}$$

3.2.4 Objective Function. The objective is to minimize the cost function, while complying with all the restrictions of the model, which is given by

$$\begin{aligned} \sum_{F \in \mathcal{F}} \sum_{t \in \mathcal{T}^F_{con}} & P^F \cdot c_t \cdot l + \sum_{t \in \mathcal{T}} \sum_{d \in \mathcal{D}} s^d_t \cdot c^d_{start} \\ & + \sum_{m \in \mathcal{M}^d_s} x^d_{t,m} \cdot c_t \cdot P^d_m \cdot l \\ & + \sum_{m \in \mathcal{M}^d_c} r^d_{t,m} \cdot c_t \cdot P^d_m \cdot l \end{aligned} \tag{14}$$

3.2.5 Discussion. In contrast to most other works on this field, we specified the model using only binary and real valued variables. Current MILP solvers benefit from this simplification due to the effectiveness of cut generation and the benefits of binary variables in branch-and-bound algorithms. Therefore, our MILP formulation can be solved rather efficiently by appropriate solvers.

The rates of a device d are preferentially modeled with a single mode and a continuous rate $r^d_{t,m}$ for each time slot t or with multiple modes and a fixed rate for each mode.

4 EVALUATION

In this section, we evaluate the effect of our proposed optimization method. To that end, we first consider a model of a cement plant from literature and real day-ahead energy prices. For comparison purposes, we compute a default schedule which is optimized for constant energy prices. We illustrate the nature of a schedule and its impact on storages. We study schedules optimized for variable energy prices, show that they reduce energy costs compared to default schedules, in particular if sufficient scheduling flexibility is available, and that they exhibit more variable storage levels. Then, we introduce optimization with look-ahead and demonstrate that it can further reduce energy costs. We also show that the savings potential depends on storage sizes and energy price variability. Finally, we report on the runtime performance of the optimization programs.

4.1 Model Description

We describe the model of the production process and the energy prices used for the optimization case study.

4.1.1 Cement Plant Model. Our evaluation is based on the cement plant model described in [1]. Its complexity is rather low. Therefore, only a subset of the features of our modeling framework needs to be leveraged. However, cement production is an energy-intensive process, for which scheduling optimization may be very helpful and effective to save energy costs.

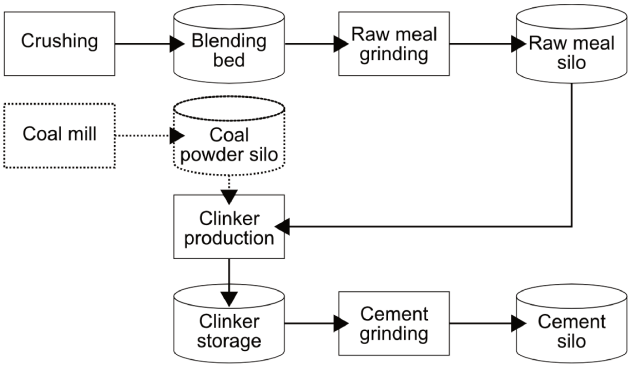


Figure 1: Process of a cement plant [1].

Table 7: Device parameters [1].

Device	Electrical demand	Material efficiency	Maximum Rate
Crusher	0.0016 MWh/t	1	200 t/h
Raw mill	0.01 MWh/t	0.8	200 t/h
Clinker production	0.017 MWh/t	1.52	95 t/h
Grinder	0.033 MWh/t	0.95	200 t/h

Table 8: Storage parameters [1].

Storage	Min. level	Max. level
Blending bed	200 t	1 800 t
Raw meal silo	200 t	1 800 t
Clinker storage	2 000 t	18 000 t
Cement silo	2 000 t	18 000 t

An overview of the cement plant model is depicted in Figure 1. The raw material is crushed and stored in a blending bed. From there it is ground by a raw mill and filled into the raw meal silo from where it is taken for clinker production. In the last step, the cement is ground from the material in the clinker silo and stored in a cement silo where it can be picked up on demand.

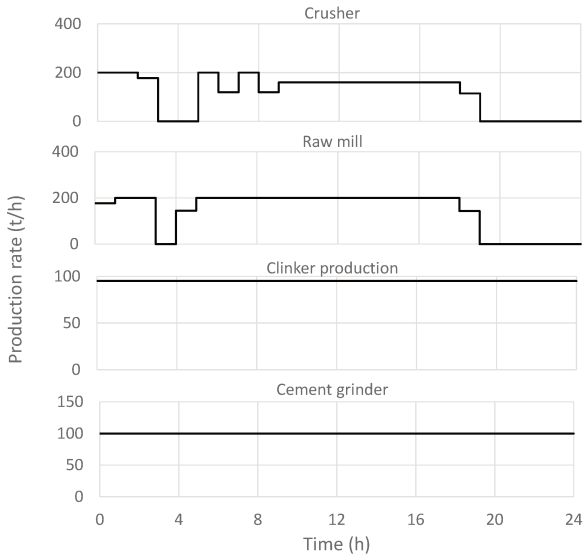
The parameters of these devices and storages are compiled in Tables 7 and 8. Table 7 shows the energy needed by a device to produce one ton of its output material. Crusher, raw mill, and grinder have variable, continuous production rates with a maximum of 200 t/h. The grinder is the most energy-intensive process followed by the clinker production. However, the latter must continuously run at its maximum rate so that it does not provide any scheduling flexibility. The raw mill and especially the crusher require substantially less energy. The table also shows the material efficiencies of the devices, i.e., the factor of how much input material is needed to produce a specific amount of output material. Instead of a production target for the entire factory, a constant cement demand of

100 t/h is specified. Detailed technical information about the coal mill and the coal powder silo are not provided in [1]. Therefore, we omit that part of the process in the evaluation.

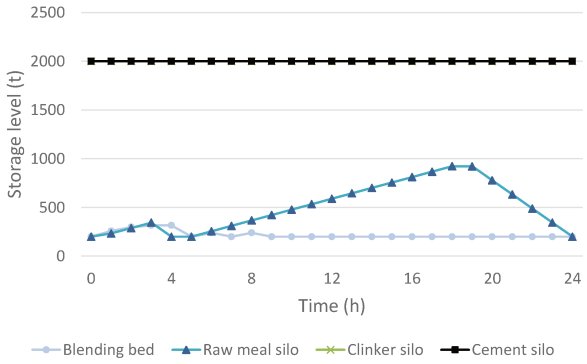
We model the cement plant using time slots with a duration of 1 hour because price forecasts are available on a hourly base. This granularity is sufficient to compute the minimum energy costs as rates are assumed to be continuous in our model. Shorter or variable-length time slots would not increase the optimization potential.

4.2 Energy Prices

Instead of price forecasts, our evaluation is based on historical day-ahead prices. We use spot market prices of the year 2018 provided by Nord Pool [12] for the day-ahead market in western Denmark (DK1).



(a) Run times visualized for crusher, raw mill, clinker production, and grinder.



(b) Time-dependent fill states of blending bed, raw meal silo, clinker silo, and cement silo.

Figure 2: Default schedule obtained from optimization for constant energy prices.

4.3 Default Schedule

We derive a default schedule which is optimized for constant energy prices. To that end, we choose a constant energy price of 30 €/MWh. For times between 7 pm and 7 am we add a penalty of 10 €/h and running device to respect increased operation costs due to shift work at night. This is only needed to obtain a schedule which preferentially runs between 7 am and 7 pm and has no impact on real energy costs. Based on this input, an optimized default schedule is computed. The real energy cost of the default schedule are computed by taking its run times with desired energy prices into account instead of constant energy prices.

The resulting run times of the devices are illustrated in Figure 2(a). The time slots are indicated on the x-axis and the height of the bars represents the production rate in the respective time slot. The crusher and the raw mill run preferentially between 7 am and 9 pm plus in the early morning hours to get the plant working. The clinker production runs permanently as this is a model-inherent requirement. The cement grinder basically works in lock step with the clinker production because the produced clinker is just enough for the demanded cement output of the factory and there is not enough material in the clinker storage so that the grinder cannot do any advance work.

Figure 2(b) shows the corresponding fill states of the storage units in the cement plant model. All minimum storage levels are met, i.e., 200 t for blending bed and raw meal silo, and 2 000 t for clinker storage and cement silo. The blending bed is mostly filled to its minimum as crusher and raw mill almost work in lock step. The raw meal silo is filled between 7 am and 7 pm and drained afterwards. As mentioned above, clinker production and cement silo are filled only to their minimum.

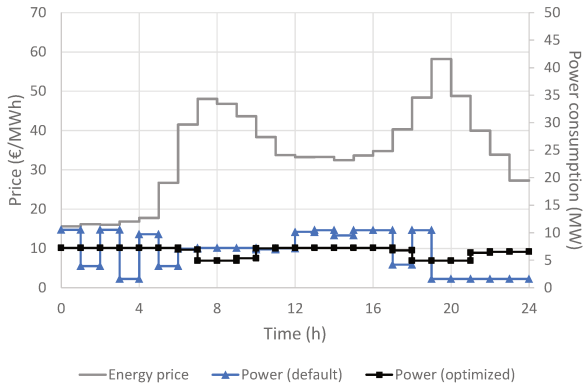
As the schedule in Figure 2(a) and the storage fill levels in Figure 2(b) are rather complex and difficult to interpret, we focus in the remainder of the paper on the fill state of the cement silo to discuss effects of different schedules. The reason for that choice is that the cement silo is filled by the grinder which is the most energy-intensive device.

4.4 Single-Day Optimization

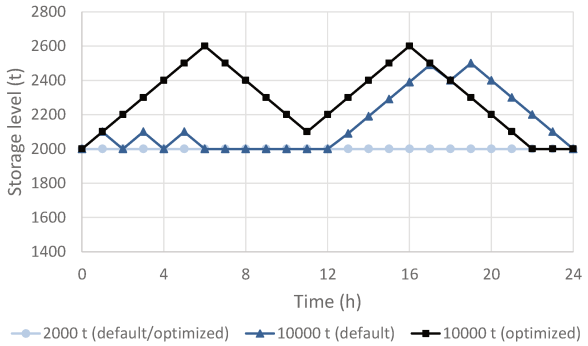
We first evaluate our optimization model for the day 2018-05-07 whose energy prices are compiled in Figure 3(a).

We presume that the plant starts with the minimum allowed fill states for all storages as indicated in the model description. We compute an optimized schedule and illustrate its effect by the fill state of the cement silo in Figure 3(b). It is exactly the same curve as the one for the default schedule. The reason for that is that the cement plant started with the minimum fill state for both the clinker storage and cement silo and that the clinker storage is constantly filled with just as much clinker as needed for the constant cement output. Therefore, both storages remain at their minimum level and the grinder cannot do any advance work but needs to work in lock step with the clinker production. As a result, there is hardly any flexibility for the schedule of the grinder. Therefore, schedule optimization is only little effective. Table 9 shows that only 3.11% of the energy costs can be saved by the optimized schedule.

It is very unfortunate that the most cost-intensive device of the production process cannot be moved in time. To avoid that



(a) Energy prices and power consumption for a start fill state of the clinker storage of 10 000 t.



(b) Storage fill states of the cement silo; the start fill state of the clinker storage is indicated; optimization is without look-ahead.

Figure 3: Investigation of various default and optimized schedules for 2018-05-07 with different start fill states for the clinker storage.

Table 9: Energy costs and savings for two different fill states of the clinker storage at start; optimization is without look-ahead.

Fill state	Schedule	Total cost	Abs. sav.	Rel. sav.
2 000 t	Default	5 365.22 €	—	—
2 000 t	Optimized	5 198.30 €	166.92 €	3.11%
10 000 t	Default	5 232.29 €	—	—
10 000 t	Optimized	4 494.37 €	737.92 €	14.10%

phenomenon, we now consider the model with a start fill state of 10 000 t for the clinker storage and compute a new default and optimized schedule. Figure 3(b) shows that the default schedule now fills the cement silo before 7 pm just as much that the grinder does not need to run anymore until the end of the day. This also reduces the energy cost for the default schedule by 2.48%. However, this improvement is rather by chance as the grinder accidentally runs at cheaper times, which may be different on another day.

Table 10: Energy costs and savings for the week from 2018-03-03 until 2018-03-09.

Schedule	Total cost	Abs. sav.	Rel. sav.
Default	46 833.14 €	—	—
Opt. w/o look-ahead	42 717.43 €	4 115.71 €	8.79%
Opt. w/ 1 day look-ahead	41 452.72 €	5 380.42 €	11.49%
Opt. w/ 2 days look-ahead	41 077.39 €	5 755.75 €	12.29%
Opt. w/ 6 days look-ahead	41 000.44 €	5 832.70 €	12.45%

In contrast, the optimized schedule intentionally leverages cheap times in the morning and in the afternoon to run the grinder filling the cement silo. Again, the fill state of the cement silo returns to its allowed minimum at the end of the day. Table 9 shows that the optimized schedule now saves 14.10% compared to the new and cheaper default schedule.

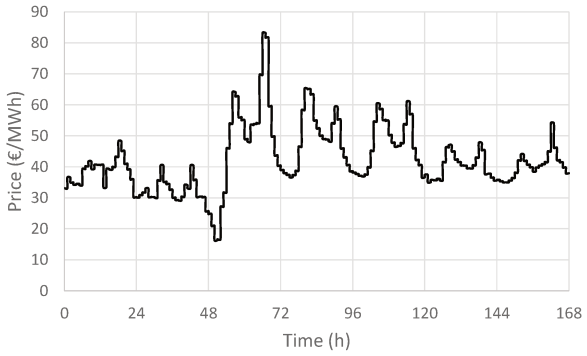
Figure 3(a) visualizes the power consumption for a start fill state of the clinker storage of 10 000 t for the default schedule and the optimized schedule. Based on this information, energy is bought at the day-ahead market. The time-dependent power consumption shows the effect of the schedule on required energy. The optimized schedule reduces power consumption when energy prices are highest, which leads to lowest total energy cost according to Table 9. However, optimization can reduce power consumption only to a certain extent during times of high energy prices as conditions like production goals and bounds for storage levels at the end of the day must be met.

4.5 Optimization with Look-Ahead

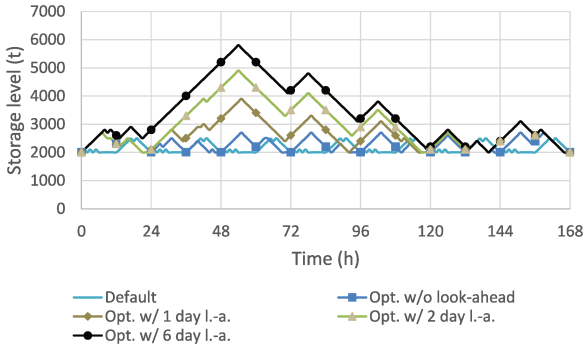
We observe that the single-day optimization leads to minimum storage fill states at the end of the day. When planning consecutive days, this implies that work cannot be done a day in advance even if price forecasts indicate more expensive prices the day after. To get rid of this artificial restriction, we introduce optimization with look-ahead. That means, when we plan the schedule for the next day, we consider n additional days for the optimization, i. e., the optimization looks ahead into the future. However, only the schedule for the next day is taken as a result and the fill states at the end of that day are used as start states for optimizing the day after using the same method. Thus, day-ahead optimization with look-ahead requires energy price forecasts for $n + 1$ days as well as predicted storage fill levels for the end of the current day.

In the following, we evaluate the benefit of optimization with look-ahead based on an interval of one week and one year, respectively. We apply the method as follows. If we apply optimization with a look-ahead of n days and there are only $m < n + 1$ days left in the interval, then we reduce the look-ahead to $m - 1$ days. This is needed to avoid unnecessarily large storage levels and energy prices at the end of the considered interval.

4.5.1 Evaluation over One Week. We evaluate optimization with look-ahead based on the energy prices of the week from 2018-03-03 until 2018-03-09 which are depicted in Figure 4(a). Figure 4(b) illustrates the fill states of the cement silo for the default schedule and for schedules optimized with a look-ahead of 0, 1, 2, and 6 days.



(a) Energy prices.



(b) Fill states of the cement silo for schedules optimized with look-ahead.

Figure 4: Investigation of the week from 2018-03-03 until 2018-03-09.

The default schedule results in periodic fill states. The schedule optimized without look-ahead also exhibits a daily pattern but with different peaks. In contrast, schedules optimized with 1 or more days look-ahead can fill storages to a larger extent and keep them filled for longer time than a single day. The maximum fill level increases with the size of the look-ahead. In all cases, the cement silo is drained to its minimum fill state at the end of the week. This is a required feature as any larger fill states would imply more energy consumed by the grinder.

Table 10 compiles the total energy costs for the week from 2018-03-03 until 2018-03-09. It shows them for the default schedule and for schedules optimized with a different numbers days look-ahead. We observe that optimization without look-ahead reduces the energy costs compared to the default schedule significantly (8.8%). Optimization with 1 or more days look-ahead reduces energy costs even further (11.5%–12.5%). However, energy price forecasts for more than 2 days are less precise, which would degrade the quality of the planning result in practice. Thus, taking only the near future into account makes the method less susceptible to forecast errors. Therefore, we consider only 1 day look-ahead from Section 4.6 on.

4.5.2 Evaluation over One Year. We now apply optimization with look-ahead to the energy prices of the entire year 2018 to assess the benefit of the method in the long run. Table 11 compiles the total energy costs for that year. We observe an energy cost reduction of about 8.0% for single-day optimization and about 10.5%–11.9% for optimization with look-ahead.

Table 11: Energy costs and savings through schedule optimization for the year 2018.

Schedule	Total cost	Abs. sav.	Rel. sav.
Default	2 524 375.50 €	—	—
Opt. w/o look-ahead	2 323 047.60 €	201 327.89 €	7.98%
Opt. w/ 1 day look-ahead	2 258 921.07 €	265 454.43 €	10.52%
Opt. w/ 2 days look-ahead	2 242 216.13 €	282 159.38 €	11.18%
Opt. w/ 6 days look-ahead	2 224 976.50 €	299 399.00 €	11.86%

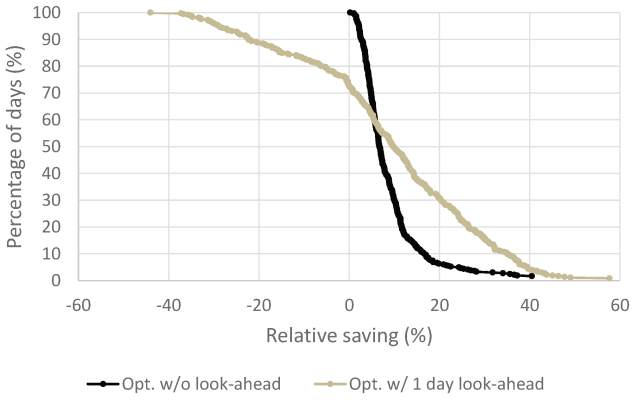


Figure 5: Daily energy savings in 2018 for schedules optimized with and without look-ahead relative to the energy costs of default schedules.

To better understand the effect of schedules optimized with and without look-ahead, we compare their energy costs to the one of default schedules on individual days. To that end, we study relative energy savings per day. Figure 5 quantifies how they are distributed for the days of the year 2018. The figure indicates the percentage of days with relative energy savings larger than a given value. For optimization without look-ahead, energy is saved on any day of the year. In 30% of the days, energy savings are larger than 10%. In contrast, for optimization with 1 day look-ahead, 28% of the days exhibit slightly larger energy costs than those of the default schedule, but 49% of the days have energy costs that are 10% cheaper than those of the default schedule. The reason for increased energy costs on some days is that work is carried out in advance due to cheap energy prices, i.e., electricity demand is shifted across day boundaries. We omit the energy savings curves for 2 and 6 days look-ahead as they are very similar to the one for 1 day look-ahead.

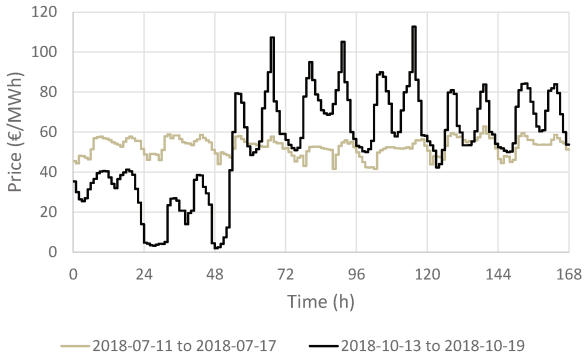
4.6 Impact of Storage Sizes

Optimization potential depends on how much energy consumption can be shifted over time. It depends on scheduling flexibility which is limited through storage sizes. To underline this proposition, we reduce the maximum fill level for all storages to the maximum values that were taken under the default schedule. Those are 320 t for the blending bed, 922 t for the raw meal silo, 10 000 t for the clinker storage, and 2 500 t for the cement silo.

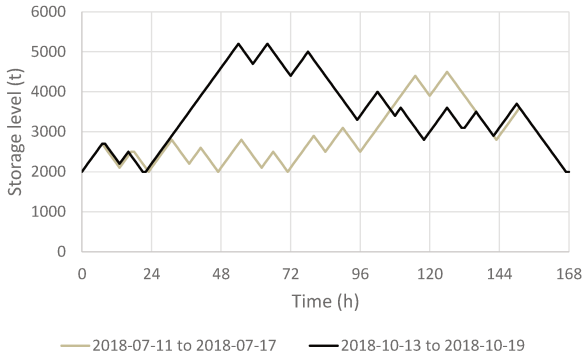
Table 12: Energy costs and savings through schedule optimization for the year 2018 depending on the storage sizes; optimization leverages a look-ahead of 1 day.

Storage	Schedule	Total cost	Abs. sav.	Rel. sav.
Standard	Default	2 524 375.50 €	—	—
Standard	Optimized	2 258 921.07 €	265 454.43 €	10.52%
Reduced	Optimized	2 333 514.02 €	190 861.48 €	7.56%

Table 12 compiles the total energy costs for 2018, for different storage sizes, and for different schedules. By construction of the experiment, reducing the storage sizes does not change the energy costs of the default schedule. For schedules optimized with 1 day look-ahead, the energy costs can be now reduced only by 7.56% for the smaller storage limits instead of 10.52% for the normal storage limits compared to the default schedule. Thus, storage sizes can significantly limit the optimization potential for energy costs.



(a) Energy prices for the weeks with the smallest and largest standard deviation in 2018: week from 2018-07-11 until 2018-07-17 (std. dev. 4.41) and week from 2018-10-13 until 2018-10-19 (std. dev 24.98).



(b) Fill states of the cement silo for both weeks for schedules optimized with 1 day look-ahead.

Figure 6: Impact of energy price variability on optimal schedules.

Table 13: Energy costs and relative savings through optimized schedules for the weeks from 2018-07-11 until 2018-07-17 (low variability, std. dev. 4.41) and from 2018-10-13 until 2018-10-19 (high variability, std. dev. 24.98); optimization leverages a look-ahead of 1 day.

Week	Schedule	Total cost	Abs. sav.	Rel. sav.
Low variability	Default	57 698.34 €	—	—
	Optimized	55 520.07 €	2 178.27 €	3.76%
High variability	Default	58 534.67 €	—	—
	Optimized	46 994.13 €	11 540.54 €	19.72%

Table 14: Computation times for different optimization intervals.

Optimization interval	Variables	Avg. computation time
1 day	652	313 ms
2 days	1 300	606 ms
3 days	1 948	1 009 ms
7 days	4 540	5 729 ms

4.7 Impact of Energy Price Variability

It is obvious that constant energy prices do not offer any potential for energy cost reduction when optimizing schedules based on energy prices. The potential for energy cost reduction obviously depends on energy price variability. To obtain an impression of the savings potential in the presence of realistic energy prices with low and high variability, we choose the weeks from 2018 with least and most variable energy prices. Their energy costs are given in Figure 6(a). The week from 2018-07-11 until 2018-07-17 has energy prices with a standard deviation of 4.41 € while the week from 2018-10-13 until 2018-10-19 has energy prices with a standard deviation of 24.98 €. We optimize schedules using 1 day look-ahead. Figure 6(b) visualizes them by the fill states of the cement silo. Both curves fluctuate as storages are filled during times of low energy cost, no matter how strong the price variability is. However, looking at Table 13, we recognize that energy costs of the default schedule can be reduced only by 3.76% for the week with little variability while they can be reduced by 19.72% for the week with high variability. Thus, if future energy prices will be more variable than today, we can expect from schedule optimization larger benefits over the year than evaluated for 2018.

4.8 Performance Considerations

The presented case study was executed on an Intel Core i5-8250U CPU @ 1.60 GHz, using 8 virtual cores and 16 GB memory. The software for constructing the models and invoking the mathematical programming solver was implemented in Java 8. We used IBM CPLEX 12.8.0.0 to solve the MILPs. The solving process needed approximately 1 GB memory for all case studies. The computation time for constructing a model and solving it using CPLEX increased with the duration of the optimization interval as the number of

variables increased. This is illustrated in Table 14. Longer look-ahead requires longer optimization intervals and more variables. However, we recommend to utilize only 1 day look-ahead as this is good tradeoff between savings improvements and precise energy price forecasts.

We point out that the presented computation times are specific to the considered case study. They may be significantly larger when the model is more complex to optimize.

5 CONCLUSION

In this work, we proposed a comprehensive framework to model an industrial plant including devices, storage units, dependencies, restrictions, and production targets for the purpose of energy cost reduction. We formulated the optimization program as a MILP. For a time series of day-ahead energy prices, the MILP computes optimal run times for the devices to minimize energy costs. To demonstrate the applicability of the proposed framework, we modeled a cement plant from the literature [1] and computed optimal schedules based on real day-ahead energy prices.

The results showed that this method works, that storages need to have appropriate fill states, and that 8% of the energy costs could be saved in 2018. We proposed optimization with look-ahead to cope with the problem that empty storages at the end of the next day may be counterproductive for the planning of the day after. It essentially extends the optimization interval but leverages only the planning for the next day which then may have non-empty storages at its end. We showed that this approach can utilize large storages to a larger extent and over a longer duration than optimization without look-ahead. It allowed improved energy cost reduction of 10.5%–11.9% in 2018, depending on the duration of the look-ahead. In addition, we showed that the optimization potential depends on storage sizes and energy price variability. The run time for the MILP was rather short, mostly below 1 second although a large number of variables were required.

From these results, we conclude that energy-intensive enterprises can save considerable energy costs using the proposed schedule optimization when purchasing energy from day-ahead markets with highly variable energy prices.

Future work encompasses the modeling and optimization of more complex plants. In particular, we will extend our model to account for time- or mode-dependent operating costs for devices which may reflect, e.g., shift work at night, and other additional costs. Additional costs can influence optimal schedules as they should lead to least overall costs. In our case study, additional costs were not taken into account to lack of information in the model from literature. Criteria to predict scheduling flexibility and optimization complexity may be helpful for efficient modeling and optimization. Furthermore, storage dimensioning and appropriate start states to leverage flexibilities for energy cost reduction may be an issue. Additionally, the robustness of the proposed solution regarding forecast errors will be investigated.

ACKNOWLEDGMENTS

The research leading to these results received funding from the German Federal Ministry for Economic Affairs and Energy under the ZIM programme (Zentrales Innovationsprogramm Mittelstand), grant no. 16KN039521. The authors alone are responsible for the content of this paper.

The authors thank Bernd Thomas and Uwe Ziegler for valuable feedback as well as Niels Schieber and Dominik Kriese for initial models and heuristic algorithms.

REFERENCES

[1] Peter Bazan, David Steber, and Reinhard German. 2017. Hybrid Simulation and Energy Market Based Optimization of Cement Plants. *Computer Science - Research and Development* 32, 1 (2017), 65–77.
[2] Pedro M. Castro, Iiro Harjunkoski, and Ignacio E. Grossmann. 2011. Optimal Scheduling of Continuous Plants with Energy Constraints. *Computers & Chemical Engineering* 35, 2 (2011), 372–387.
[3] Christian Gahm, Florian Denz, Martin Dirr, and Axel Tuma. 2016. Energy-efficient scheduling in manufacturing companies: A review and research framework. *European Journal of Operational Research* 248, 3 (2016), 744–757.
[4] Florian Heimgaertner, Uwe Ziegler, Bernd Thomas, and Michael Menth. 2018. A Distributed Control Architecture for a Loosely Coupled Virtual Power Plant. In *Proceedings of the ICE/IEEE International Technology Management Conference (ICE/IEEE ITMC)*.
[5] Marianthi G. Ierapetritou and Christodoulos A. Floudas. 1998. Effective Continuous-Time Formulation for Short-Term Scheduling. 2. Continuous and semicontinuous Processes. *Industrial & engineering chemistry research* 37, 11 (1998), 4360–4374.
[6] Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Proceedings of a symposium on the Complexity of Computer Computations*. Yorktown Heights, New York, USA, 85–103.
[7] Emilia Kondili, Nilay Shah, and Constantinos C. Pantelides. 1993. Production Planning for the Rational Use of Energy in Multiproduct Continuous Plants. *Computers & Chemical Engineering* 17 (1993), S123–S128.
[8] Naseer Abboodi Madlool, Rahman Saidur, M. Shouquat Hossain, and Nasrudin Abd Rahim. 2011. A Critical Review on Energy Use and Savings in the Cement Industries. *Renewable and Sustainable Energy Reviews* 15, 4 (2011), 2042–2060.
[9] Carlos Alberto Mendez and Jaime Cerda. 2002. An Efficient MILP Continuous-Time Formulation for Short-Term Scheduling of Multiproduct Continuous Facilities. *Computers & Chemical Engineering* 26, 4-5 (2002), 687–695.
[10] Sumit Mitra, Ignacio E. Grossmann, Jose M. Pinto, and Nikhil Arora. 2012. Optimal Production Planning under Time-Sensitive Electricity Prices for Continuous Power-Intensive Processes. *Computers & Chemical Engineering* 38 (2012), 171–184.
[11] Klaus Neumann, Christoph Schwindt, and Norbert Trautmann. 2005. Scheduling of Continuous and Discontinuous Material Flows with Intermediate Storage Restrictions. *European Journal of Operational Research* 165, 2 (2005), 495–509.
[12] Nord Pool AS. 2018. Historical Market Data. https://www.nordpoolgroup.com/historical-market-data/.
[13] Munawar A Shaik and Christodoulos A Floudas. 2007. Improved Unit-Specific Event-Based Continuous-Time Model for Short-Term Scheduling of Continuous Processes: Rigorous Treatment of Storage Requirements. *Industrial & engineering chemistry research* 46, 6 (2007), 1764–1779.
[14] Fadi Shrouf, Joaquin Ordieres-Meré, Alvaro García-Sánchez, and Miguel Ortega-Mier. 2014. Optimizing the Production Scheduling of a Single Machine to Minimize Total Energy Consumption Costs. *Journal of Cleaner Production* 67 (2014), 197–207.

APPENDIX

A PROOF OF NP-HARDNESS

To proof the NP-hardness of the considered scheduling problem, it must first be formulated as a decision problem.

Definition 1 (Energy-Scheduling).
Given: *A system of devices, storages, fixed consumers, and restrictions, defined as in Section 3, given by the respective parameters, energy prices for the entire planning horizon, and maximal cost* $C \in \mathbb{Z}$.
Question: *Is there a schedule which fullfils all restrictions imposed by the given model while inducing costs of at most* C*?*

The following problem is also needed for our proof.

Definition 2 (Knapsack).
Given: *Objects* $(w_1, v_1), ..., (w_n, v_n)$*, consisting of weight and value,*

a maximum weight $W \in \mathbb{N}$ and a minimum value $N \in \mathbb{N}$
Question: *Is it possible to choose a subset of objects such that the sum of the weights of its elements does not exceed W while the sum of the values of its elements is at least N?*

Knapsack is a well known NP-complete problem, a fact which was first proven by Karp [6].

Theorem A.1. *Energy-Scheduling is NP-hard.*

Proof. By giving a polynomial-time reduction from Knapsack to Energy-Scheduling, NP-completeness of Energy-Scheduling can be proven. Let $(w_1, v_1), ..., (w_n, v_n)$ be an instance of Knapsack. Construct one device per object (w_i, v_i). This device has only one continuous mode. The maximum rate of this mode is normalized such that the device will produce v_i units of end products if the device runs for the entire optimization interval. The energy consumption of the mode is also normalized such that w_i units of energy are consumed if the device runs for the entire optimization interval. The minimal production per run is set to v_i. All these devices are connected to one common storage. The production target of this storage is set to N. Energy prices are set to one unit during the entire optimization interval. The maximum cost C is set to W.

If there is a subset of objects which fulfills the requirements of Knapsack, there is a schedule for the constructed model with costs of at most W. Such a schedule can be constructed by letting work the respective devices of the objects contained in the subset for the entire optimization interval. All other devices do not work at all. Through the normalization of production rates corresponding to the values of the respective objects, the production target of the common storage is fulfilled. By the same argument, the cost induced by this schedule is at most W.

For the contrary, suppose there is no subset of objects with the needed requirement, but there is a schedule for the constructed model which fulfills the production target and maximum cost restriction. It is implied by the minimal production per run that a device can only work for the whole planning horizon or not at all. By taking the corresponding objects of the running devices in this schedule, one gets due to the normalization of rates and energy demands a subset of objects with a sum of weights of at most W and a sum of values of at least N. This contradicts the assumption that there is no such subset, so there cannot be such a schedule for the constructed model. To see the polynomial run time of this construction, observe that only one device is constructed per object with one additional common storage and C is just a copy of W. So the construction is indeed a polynomial-time reduction from Knapsack to Energy-Scheduling, which completes the proof of NP-hardness. □

If there is an algorithm which computes the optimal schedule in polynomial time, it could be used to decide Energy-Scheduling, which would imply the commonly as unlikely seen statement of $P = NP$.

B MILP-REPRESENTATION OF ADDITIONAL CONSTRAINTS

Auxiliary parts of the MILP are presented in this section. They enforce the intended semantics of the variables presented in Table 2. In every valid assignment of variables, for a device d must hold that start- and end-of-run variables, which are set to 1, must alternate. Additionally, the first of these variables, which is set to 1, must be a start-of-run variable while the last one has to be an end-of-run variable. The following inequalities implement these restrictions.

$$\forall t \in \mathcal{T} : 0 \leq \sum_{t' \in \mathcal{T},\ t' \leq t} s^d_{t'} - \sum_{t' \in \mathcal{T},\ t' < t} e^d_{t'} \leq 1 \tag{15}$$

$$\forall t \in \mathcal{T} : \sum_{t' \in \mathcal{T},\ t' \leq t} e^d_{t'} \leq \sum_{t' \in \mathcal{T},\ t' \leq t} s^d_{t'} \tag{16}$$

$$\forall t \in \mathcal{T} : 1 + \sum_{t' \in \mathcal{T},\ t' \leq t} e^d_{t'} \geq \sum_{t' \in \mathcal{T},\ t' \leq t} s^d_{t'} \tag{17}$$

$$\sum_{t' \in \mathcal{T}} e^d_{t'} = \sum_{t' \in \mathcal{T}} s^d_{t'} \tag{18}$$

That a device can only run in at most one mode in every slot is modeled by

$$\forall t \in \mathcal{T} : \sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{t,m} \leq 1. \tag{19}$$

The semantics of the run variables for every slot demand that they are only set to 1 if and only if there is a start-of-run variable set to 1 in an earlier slot and no end-of-run variable in a slot between. Because at most one of the run variables of a single device in a given slot can be set to 1, the sum of these variables can be understood as a single binary variable itself.

$$\forall t \in \mathcal{T} : s^d_t \leq \sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{t,m} \tag{20}$$

$$\forall t \in \mathcal{T} \setminus \{1\} : -e_{t-1} + \sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{t-1,m} \leq \sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{t,m} \tag{21}$$

$$\forall t \in \mathcal{T} \setminus \{1\} : -s_t - \sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{t-1,m} \leq \sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{t,m} \tag{22}$$

$$\forall t \in \mathcal{T} \setminus \{1\} : e_{t-1} - s_t \leq 1 - \sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{t,m} \tag{23}$$

$$\sum_{m \in \mathcal{M}_s \cup \mathcal{M}_c} x^d_{1,m} \leq s^d_1 \tag{24}$$

At last, the semantics of the cumulative-production variables need to grow over a run dependent on the production rate in every time slot and should be set to 0 when a run ends. After the first slot in the optimization interval, the cumulative variable of every device should be initialized with the production of the respective device in the first slot. Let $s \in O(A)$ be an output of the device d.

$$\forall t \in \mathcal{T} \setminus \{1\} : k^d_{t,s} - l \cdot \left(\sum_{m \in \mathcal{M}_s} m(s) \cdot x^d_{t,m} + \sum_{m \in \mathcal{M}_s} r^d_{t,m} \right) \leq n \cdot l \cdot Max^d \cdot (1 - e^d_{t-1}) \tag{25}$$

$$\forall t \in \mathcal{T} \setminus \{1\} : l \cdot \left(\sum_{m \in \mathcal{M}_s} m(s) \cdot x^d_{t,m} + \sum_{m \in \mathcal{M}_c} r^d_{t,m} \right) - k^d_{t,s} \leq n \cdot l \cdot Max^d \cdot (1 - e^d_{t-1}) \tag{26}$$

$$\forall t \in \mathcal{T} \setminus \{1\} : \left(k^d_{t-1,s} + l \cdot \left(\sum_{m \in \mathcal{M}_s} m(s) \cdot x^d_{t,m} + \sum_{m \in \mathcal{M}_c} r^d_{t,m} \right) \right) - k^d_{t,s} \leq n \cdot l \cdot Max^d \cdot e^d_{t-1} \tag{27}$$

$$\forall t \in \mathcal{T} \setminus \{1\} : k^d_{t,s} - \left(k^d_{t-1,s} + l \cdot \left(\sum_{m \in \mathcal{M}_s} m(s) \cdot x^d_{t,m} + \sum_{m \in \mathcal{M}_c} r^d_{t,m} \right) \right) \leq n \cdot l \cdot Max^d \cdot e^d_{t-1} \tag{28}$$

$$\sum_{m \in \mathcal{M}_s} l \cdot m(s) \cdot x^d_{1,m} + \sum_{m \in \mathcal{M}_c} l \cdot r^d_{1,m} = k^d_{1,s} \tag{29}$$

For continuous modes, it must be enforced that the continuous rate variable is 0 if and only if the run variable is set to 0 for the respective mode in all time slots. Additionally, the rate must be within the respective bounds of the mode.

$$\forall t \in \mathcal{T} \forall m \in \mathcal{M}^d_c : r^d_{t,m} \leq m_{max} \tag{30}$$

$$\forall t \in \mathcal{T} \forall m \in \mathcal{M}^d_c : -r^d_{t,m} + m_{min} \leq m_{max} - (m_{max} \cdot x^d_{t,m}) \tag{31}$$

$$\forall t \in \mathcal{T} \forall m \in \mathcal{M}^d_c : 0 \leq r^d_{t,m} \leq m_{max} \tag{32}$$

1.2 A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN)

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 11 May 2023, accepted 9 June 2023, date of publication 14 June 2023, date of current version 22 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3286370

SURVEY

A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN)

THOMAS STÜBER, LUKAS OSSWALD, STEFFEN LINDNER, AND MICHAEL MENTH, (Senior Member, IEEE)
Chair of Communication Networks, University of Tübingen, 72074 Tübingen, Germany

Corresponding author: Thomas Stüber (thomas.stueber@uni-tuebingen.de)

This work was supported by the German Federal Ministry of Education and Research (BMBF) through the Collaborative Project Künstliche Intelligenz zur dynamischen Optimierung des Netzwerkmanagements (KITOS) under Grant 16KIS1161. We additionally acknowledge the support from the Open Access Publication Fund of the University of Tübingen.

ABSTRACT Time-Sensitive Networking (TSN) is an enhancement of Ethernet which provides various mechanisms for real-time communication. Time-triggered (TT) traffic represents periodic data streams with strict real-time requirements. Amongst others, TSN supports scheduled transmission of TT streams, i.e., the transmission of their frames by end stations is coordinated in such a way that none or very little queuing delay occurs in intermediate nodes. TSN supports multiple priority queues per egress port. The TAS uses so-called gates to explicitly allow and block these queues for transmission on a short periodic timescale. The TAS is utilized to protect scheduled traffic from other traffic to minimize its queuing delay. In this work, we consider scheduling in TSN which comprises the computation of periodic transmission instants at end stations and the periodic opening and closing of queue gates. In this paper, we first give a brief overview of TSN features and standards. We state the TSN scheduling problem and explain common extensions which also include optimization problems. We review scheduling and optimization methods that have been used in this context. Then, the contribution of currently available research work is surveyed. We extract and compile optimization objectives, solved problem instances, and evaluation results. Research domains are identified, and specific contributions are analyzed. Finally, we discuss potential research directions and open problems.

INDEX TERMS Time-sensitive networking (TSN), time-aware shaper (TAS), scheduling, optimization, ethernet bridging.

I. INTRODUCTION

Modern applications, e.g., Industry 4.0 factory automation and motion control, demand highly deterministic network service. Exceeding latency and jitter bounds can result in immediate degradation of manufacturing quality or endanger health of machinery and operators. Some of these applications have to exchange data streams with precise timing to keep application-specific deadlines. Time-Sensitive Networking (TSN) is an emerging technology which enhances Ethernet networks with real-time properties. In TSN, talkers send uni- or multicast streams, called streams, to traffic sinks, called listeners. The network admits streams and guarantees quality of service (QoS). Time-triggered (TT) traffic constitutes periodic data streams with real-time requirements such as bounded latency or jitter. The transmission times of TT streams at their respective talkers must be scheduled such that excessive queuing in the network is avoided and their requirements are met. Although TT traffic has high priority, it can be delayed by low-priority frames in transmission blocking links for short time. To ensure that links are not occupied by low-priority traffic when needed for TT traffic, the standard IEEE Std 802.1Qbv [6] introduces an enhancement for scheduled traffic. The Time-Aware Shaper (TAS) can be implemented with this enhancement. It defines periodic time slices during which queues may send traffic to an output port and delays

The associate editor coordinating the review of this manuscript and approving it for publication was Divanilson Rodrigo Campelo.

TABLE 1. Surveys covering related topics to this paper.

Paper	Date	References	Focus	# Common referenced papers with this survey
Nasrallah *et al.* [1]	2018	407	Overview of TSN, DetNet, and 5G standards	14
Minaeva *et al.* [2]	2021	126	Scheduling in periodic systems	6
Seol *et al.* [3]	2021	207	Broad overview of TSN	29
Deng *et al.* [4]	2022	128	Broad overview of all topics related to AVB and TSN	17
Gavriluţ *et al.* [5]	2023	88	History of real-time Ethernet technologies, problem statements and solving techniques for various network design problems	13
This paper	2022	139	Traffic scheduling in TSN with the TAS	-

the respective traffic. In TSN, the TAS is used to protect TT traffic from other traffic classes. Therefore, TSN requires that appropriate TAS time slices are scheduled for output queues on all switches, in addition to the transmission times of all TT streams at their talkers. This combination guarantees very short delays for TT streams in TSN.

Standardization does not yet cover methods for computing such schedules. However, the topic has been examined by many publications. These research works use different methods for schedule synthesis, evaluation, and objectives for optimization. We survey the currently available literature for TSN schedule computation. The paper focuses on publications published until March of 2023 about TSN schedule planning with the TAS. Works about stream scheduling related to other technologies than TSN or for other traffic shapers in TSN than the TAS are not covered in this survey.

A. RELATED SURVEYS

To the best of our knowledge, no other review covers scheduling algorithms for TSN as its main topic. In fact, there is no survey about scheduling for TT streams for Ethernet networks, regardless of the used standard. However, there are surveys which intersect with the content of this work. Table 1 compiles the focus and the relationship of these surveys to this paper.

Nasrallah et al. [1] survey standards for low-latency communication. Besides DetNet and 5G, they also give a tutorial on the TSN standards. They reference a small number of papers related to traffic scheduling in TSN. However, they do not elaborate on their content as scheduling is not the focus of this work. Thus, only the most seminal works about scheduling from this time are referenced.

Minaeva et al. [2] give a literature summary for scheduling time-triggered real-time systems. They highlight research works from 1968 to 2020. As opposed to this work, they not only consider scheduling of streams in networks, but all systems with periodic schedules. Seminal works for TSN scheduling algorithms in the literature are mentioned, e.g., [7] and [8]. Out of 126 references, only 6 of them intersect with this work.

Deng et al. [4] review a wide range of topics about AVB and TSN from the literature of 2007 – 2021. Besides scheduling approaches, they also give an overview of reliability and security modeling, and delay analysis in the mentioned areas. As a wider range of topics is covered, only a small part of the survey is concerned with scheduling. From the 128 discussed works, 17 works intersect with this survey.

Seol et al. [3] review TSN as a whole. The authors cover publications of the years 2014 – 2020. An overview of active research directions is given, including computing routings and schedules in TSN. Not only the literature about scheduling for the TAS is summarized, but also work concerned with other queuing mechanisms, hardware, and simulation frameworks. Therefore, only a small fraction of the literature about scheduling for TAS-based queuing in TSN is surveyed. They cover 207 research works, of which 29 are included in this work. Because of the wide range of topics covered, these works are referenced for further reading but their content is not discussed.

The recent survey of Gavriluţ et al. [5] gives an excellent introduction in the history of real-time Ethernet technologies. Additionally, they present typical problems in the design of networks for time-critical applications, e.g., scheduling, routing, worst-case delay analysis, topology synthesis, and bandwidth allocation. Seminal works for each of these problems are reported and summarized. Important results are recalled. However, the focus is much broader than the scheduling problem for the TAS. Thus, many works about scheduling were not covered.

B. CONTRIBUTION

In contrast to the mentioned surveys of Table 1, we focus on papers about scheduling algorithms and related topics which use the TAS. This survey claims the following contributions:

- We give a tutorial on TSN basics.
- We define the TSN scheduling problem for TAS and modifications to it. Additionally, we introduce common solution methods used in the literature
- We survey currently available TSN literature about scheduling for the TAS.
- We identify research directions, categorize the available literature, and highlight contributions to these topics.
- We compare the available algorithms and the presented evaluations to derive open research questions in this area.

C. SURVEY STRUCTURE

This paper is structured as follows. In Section II we present a brief introduction to TSN with a special focus on the TAS. Then, we formally define the scheduling problem in TSN and give a tutorial to common solutions methods from literature in Section III. Section IV gives an overview of the state-of-the-art of TSN scheduling and categorizes the presented literature. Section V compares the presented research work with regard to modelling assumptions, optimization objective, problem instances and scalability. Furthermore, we present the publication history of the surveyed literature in Section VI. We discuss issues and open research questions in Sections VII. Finally, we conclude the paper in Section VIII.

D. LIST OF FREQUENTLY USED ACRONYMS

The following acronyms are used in this paper.

ASAP	As Soon As Possible.
AVB	Audio Video Bridging.
BE	Best Effort.
CBS	Credit-Based Shaper.
CP	Constraint Programming.
CQF	Cyclic Queuing and Forwarding.
GA	Genetic Algorithm.
GCL	Gate Control List.
FIFO	First-In-First-Out.
FRER	Frame Replication and Elimination for Reliability.
gPTP	generalized Precision Time Protocol.
GRASP	Greedy Randomized Adaptive Search Procedure.
ILP	Integer Linear Programming.
OMT	Optimization Modulo Theories.
PBO	Pseudo-Boolean Optimization.
PSFP	Per-Stream Filtering and Policing.
QoS	Quality of Service.
SMT	Satisfiability Modulo Theories.
SRP	Stream Reservation Protocol.
TAS	Time-Aware Shaper.
TSN	Time-Sensitive Networking.
TT	Time-Triggered.
VLAN	Virtual LAN.

II. FOUNDATIONS OF TSN

TSN is a set of standards for deterministic data transmission with real-time requirements over Ethernet networks. In this section, we present a short tutorial about TSN. First, we present AVB based on which TSN was developed. Then, we introduce TSN with a special focus on scheduling and the TAS.

A. AUDIO VIDEO BRIDGING

Historically, multimedia equipment was interconnected with half-duplex point-to-point links for data transmission. These links were often dedicated to a single purpose, i.e., the transmission of one specific data stream. This results in a large number of links which is expensive, hard to maintain, and error prone. Switched computer networks solved these problems. The most widely adopted technology for switched local area networks today is Ethernet. However, professional audio and video applications need bounded latencies and jitter, i.e., real-time guarantees for data streams. Switching in Ethernet networks was not designed for real-time transmissions. Therefore, the Audio Video Bridging (AVB) task group of the IEEE was founded to develop a standard to meet the requirements of multimedia applications in switched Ethernet networks.

AVB is organized in standards for time synchronization, admission control, and traffic shaping.

1) TIME SYNCHRONISATION

Network devices need a common understanding of time to ensure that all end stations in a network are able to coordinate their actions. Every AVB-capable device is equipped with a clock. The standard IEEE 802.1AS [9] defines a protocol to synchronize the clocks of all devices in an AVB network. This protocol is based on the *Precise Time Protocol* (PTP) introduced in IEEE 1588 [10] and is denoted as *generalized Precise Time Protocol* (gPTP). The gPTP defines an algorithm to select a so-called *Grandmaster* among the participating nodes of the protocol. The internal clock of the Grandmaster is used as reference clock. All other devices synchronize their clocks to the clock of the Grandmaster with time information sent from the Grandmaster. Intermediate nodes adjust the received time information to compensate propagation delays, processing delays, and different clock speeds before retransmitting them. The gPTP allows submicrosecond precision for devices with at most seven hops distance to each other. This is needed for applications running on different end stations to synchronize their actions.

2) ADMISSION CONTROL

The Stream Reservation Protocol (SRP) introduced in IEEE 802.1Qat [11] allows senders of periodic data streams, denoted as *talkers*, to reserve bandwidth in a multi-hop Ethernet network. A talker which wants to send data advertises a new data stream to its connected bridge. This advertisement contains information about bandwidth and real-time requirements, the periodicity of the stream, and the destination MAC address. The destination may be a multicast group. The bridge forwards the advertisement if the requested resources are available. Worst-case latencies are calculated at every bridge. When the request reaches the destination of a data stream, denoted as *listener*, the listener acknowledges that it is ready, and the bandwidth is reserved along the path.

3) TRAFFIC SHAPING

Traffic shaping is the generic term for techniques that distribute packet transmissions in time. The AVB working group defines the so-called *Credit-Based Shaper* (CBS) in IEEE 802.1Qav [12]. It can be leveraged to smooth out bursts such

that receiving devices are not overwhelmed. This reduces buffering and congestion in the network. The CBS is a leaky bucket traffic shaper with at least two FIFO queues for two traffic classes. These classes are denoted as class A and class B. Both queues have a credit measured in bit. Dispatching and transmitting a frame from a queue is only allowed if the credit of the respective queue is non-negative. Credit increases linearly during times no frame is transmitted and decreases linearly during transmissions. Latency bounds for streams can be guaranteed by using a special configuration for the CBS defined in IEEE 802.1BA [13]. These are specific to the requirements of the AVB domain, guaranteeing 2 ms and 50 ms for class A and B traffic in networks with at most 7 hops. However, the average delay of a frame increases to up to 250 μs per hop in the worst case when the CBS is used.

B. TIME-SENSITIVE NETWORKING

Ethernet networks are used in a wide range of industrial use cases as Ethernet is cheap and easy to implement. However, use cases such as industrial automation, in-vehicle communication or avionics have hard real-time requirements and need reliability. Data streams not meeting their deadlines may not only be worthless but impose safety risks. The latency guarantees and average delays offered by AVB fail to comply with the requirements of such use cases.

Time-Sensitive Networking (TSN) is a set of standards enhancing AVB for deterministic and reliable transmission of data over switched Ethernet networks. TSN is currently developed in the IEEE 802.1 TSN task group and adds new mechanisms for scheduling, traffic shaping, path selection, stream reservation, filtering and policing, and fault-tolerance. Most of the standards are enhancements of IEEE 802.1Q [14] which defines bridges and Virtual LANs (VLANs). We give a brief tutorial on the standards and mechanisms relevant for the scope of this survey, i.e., traffic scheduling in TSN with the TAS.

1) SIMILARITIES TO AVB

Similar to AVB, every device in TSN is equipped with a clock. TSN also uses the gPTP defined in IEEE 802.1AS [9] to synchronize clocks of all network devices. The CBS and an enhancement of the SRP are also part of TSN.

2) PATH SELECTION

TSN introduces a new mechanism for path selection in IEEE 802.1Qca [15]. In contrast to traditional Ethernet networks, it is not necessary to use Spanning Tree Protocols or Shortest Path Bridging. Paths can be computed by an arbitrary algorithm and are only limited to be trees. Thus, frames can be forwarded on an arbitrary path. Forwarding information of these so-called Explicit Trees are distributed with the Intermediate System to Intermediate System (IS-IS) protocol and stored in bridges. The Explicit Tree for the forwarding of a frame is determined by the MAC address of the root bridge of the Explicit Tree and the VLAN ID in the frame's header.

3) PRIORITIES

Every egress port of a TSN bridge is equipped with up to eight egress queues. These queues are First-In-First-Out (FIFO) queues. They correspond to the eight VLAN priorities defined in IEEE 802.1Q [14]. The VLAN tag in the header of an Ethernet frame determines the egress queue in which the frame waits for transmission. Every queue is equipped with a so-called Transmission Selection Algorithm (TSA). The TSA signals whether a frame is ready for transmission to a transmission selection mechanism. A possible implementation for a TSA is the CBS which allows frame transmissions only when the credit is positive. This selection mechanism selects the next queue from which a frame is dispatched and sent. TSN uses strict priority as transmission selection, i.e., the next frame is dispatched from the highest priority queue which signals a frame is ready for transmission.

4) FRAME PREEMPTION

High-priority traffic can be delayed due to conflicts with lower-priority traffic. IEEE 802.1Qbu [16] describes a mechanism for frame preemption in TSN which reduces such delays. Traffic is divided into preemptable frames and so-called express frames. The transmission of preemptable frames is paused and finished later if an express frame is ready for transmission. Consequently, a preempted frame is divided into fragments which are reassembled by the receiving node. The minimum size of a frame fragment is defined to be 64 byte. However, every fragment of a frame except for the last one has a trailer containing a 4 byte check sequence for error detection. Therefore, a frame can only be preempted after at least 60 byte were transmitted and the last 63 byte of a frame cannot be preempted.

5) RELIABILITY AND THE FILTERING OF DUPLICATES

Bridging in classical Ethernet networks assumes that no frames are duplicated and therefore no duplicates must be filtered. However, safety critical applications may require protection against frame loss and permanent link failures. IEEE 802.1CB [17] introduces a mechanism which allows to sent multiple copies of the same frame, possibly over disjoint paths, and to eliminate duplicates. Thus, only a single copy of the same frame is forwarded or delivered to a higher layer on an end station. This mechanism is denoted as Frame Replication and Elimination for Reliability (FRER).

6) TRAFFIC SCHEDULING

Time-triggered (TT) traffic, also denoted as *scheduled traffic*, consists of periodic data streams with hard real-time requirements such as bounded latency and jitter. The properties of TT streams such as period, maximum frame size, frames per period, as well as the range of possible transmission offsets from their respective talkers, are known in advance. The transmission times of these streams at their respective talkers can be controlled and must be coordinated to ensure that all streams meet their real-time requirements. The computation

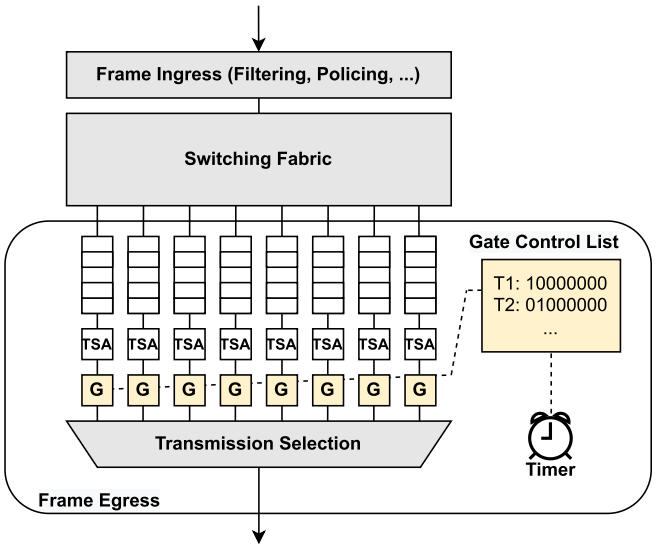


FIGURE 1. Path of a frame through a bridge. The egress port implements the enhancement for scheduled traffic. Every such egress ports has eight egress queues guarded by a transmission gate (G). The GCL controls the timed opening and closing of these gates.

of their periodic transmission times is denoted as *traffic scheduling*.

a: TRAFFIC SHAPING

TSN introduces new traffic shapers in addition to the CBS. An example for another shaper which can be used instead of the CBS is Cyclic Queuing and Forwarding (CQF) defined in IEEE 802.1Qch [18]. Time is divided into slots of predefined length. The length of a slot is denoted as cycle time. Bridges buffer all frames received during a slot and transmit them in the subsequent slot. Thus, stream latencies can easily be calculated from the cycle time and the number of hops.

IEEE 802.1Qbv [6] defines an enhancement for scheduled traffic. It can be leveraged to implement the Time-Aware Shaper (TAS). The TAS allows protecting TT traffic from other traffic such as AVB traffic or best-effort (BE) traffic. Additionally, the transit of TT streams through a network can be scheduled. Every egress queue has a so-called *transmission gate* or simply *gate*. Gates are either open or closed. Frames can only be dispatched and sent from an egress queue if the respective gate is open. The closing and opening of a gate is controlled by a so-called Gate Control List (GCL). A GCL entry consists of a time interval $[T_i, T_{i+1}]$ and a bit-vector. The bit-vector indicates which gates are opened or closed during the time interval $[T_i, T_{i+1}]$. Therefore, a GCL entries defines a time slice exclusively available to traffic with a priority corresponding to an open queue. These GCLs are executed periodically for an indefinite number of times. The computation of GCLs and appropriate cycle times, i.e., periods of these GCLs, is denoted as *scheduling* or *GCL synthesis*. The number of available GCL entries in an egress port is limited and depends on the used bridge. Figure 1 depicts the architecture of a typical TSN bridge according to IEEE 802.1Q [14], including the components of the TAS.

The TAS can be used to protect traffic by scheduling the GCLs accordingly.

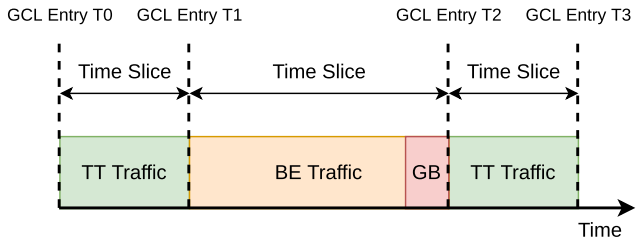


FIGURE 2. Time slices, GCL entries, and guard bands. The duration of a guard band may be not available for BE traffic as a frame can only be sent if transmission finishes before the respective gate is closed.

b: GATE CLOSINGS AND GUARD BANDS

If the transmission of a frame is not finished until the end of the time slice the transmission started, a frame in the next time slice may be forced to wait until transmission finishes. Thus, it would be possible that a frame of a TT stream must wait because of a frame of BE traffic. This problem is avoided in TSN. Bridges detect automatically whether a frame transmission would conflict with a gate closing and hold conflicting frames back in this case. A guard band is a time interval with the length of the transmission of a maximum sized standard Ethernet frame. The duration of a guard band at the end of a time slice may not be available for transmissions to comply with closed gates. However, we emphasise that guard bands in TSN are implicit, i.e., they must not be configured explicitly. Transmissions may even start during a guard band if the transmission finishes before the next gate closing. Figure 2 depicts a guard band which restricts the transmission of BE traffic before the respective gate is closed. If frame preemption is used, the maximum size of a frame that cannot be preempted is 123 byte. This is due to the minimum size of a frame fragment, i.e., 60 byte of the frame and an additional 4 byte check sequence. A frame with 123 byte cannot be preempted until the first 60 byte are transmitted as the resulting first fragment would be too small otherwise. However, the last 63 byte also cannot be preempted as the resulting last fragment would be too small. Therefore, guard bands can be reduced to the length of a transmission of 123 byte if frame preemption is used.

c: SCHEDULER VS. TRAFFIC SCHEDULING

The term *scheduler* is sometimes used as a synonym for *traffic shaper*. For instance, the CBS and the TAS are schedulers in this terminology. Unfortunately, the term *scheduler* has also another meaning in the context of this survey. Many research works denote algorithms to plan GCL entries and frame transmissions in time with the TAS as schedulers. To avoid confusions, we will only use the second meaning in the remainder of this paper, i.e., a scheduler is a scheduling algorithm for the TAS. There are research works that use

the first meaning in their title or abstract or cover scheduling algorithms for other traffic shapers in TSN, e.g., Cyclic Queuing and Forwarding (CQF). Thus, these works give the impression that they are in the scope of this survey, e.g., [19], [20], [21], [22], [23], [24], [25], [26], and [27]. However, we remark that this survey only covers research works about scheduling algorithms and the scheduling problem for the TAS. Therefore, we do not discuss works which propose new shapers, i.e., new *schedulers*, or analyse other shapers than the TAS.

The challenge in planning a TSN network is to compute the schedules that coordinate the transmission times for all streams at their respective talkers and the GCLs of bridges such that the requested real-time requirements of all TT streams are met. This problem is formally defined in the next section.

III. THE TSN SCHEDULING PROBLEM

First, we introduce a common network model, relevant properties of TT streams, and the definition of schedules. Second, we state constraints for valid schedules. Then, we discuss scheduling and optimization in the context of TSN and the computational complexity of these problems. Furthermore, we present common problem extensions solved in the literature. Finally, we give an introduction to common solution techniques that have been applied to the scheduling problem.

A. NOMENCLATURE

A node of a TSN network is an end station or a TSN-capable bridge. End stations are sources and destinations of data streams. Bridges switch frames based on their header. We remark that a node may be a bridge and an end station at the same time, i.e., it implements bridging capabilities and is an end point of data streams. Links are full-duplex Ethernet connections between an end station and a bridge or between two bridges. TSN bridges are inevitably subject to multiple delays. These delays must be considered to ensure deterministic transmissions according to a schedule. The processing delay of a bridge is the time between a frame arrives at an ingress port, and it is put in an egress queue. The transmission rate of an egress port is the rate at which data can be transmitted over a link. The propagation delay of a link is the time needed for electrical signals to traverse the link. The queuing delay of a frame is the time the frame waits in an egress queue for transmission. Ethernet uses a preamble before of a frame transmission to signal a new transmission starts, and an inter-frame gap between two frame transmissions to ensure that the receiver can process a new frame. The maximum size of a frame in TSN is 1542 byte, including inter-frame gap and preamble. A TT stream is a periodically repeated data stream with real-time requirements. Every stream has a talker as source and possibly multiple listeners as destinations. The earliest and latest transmission offsets describe the time range during which a talker can start transmission relative to the start of a period. The deadline of a stream is the time at which all frames of the stream must have arrived at all destinations, also relative to the start of a period. The entire payload of a stream must be delivered before the deadline. The payload of a stream may be sent with multiple frames.

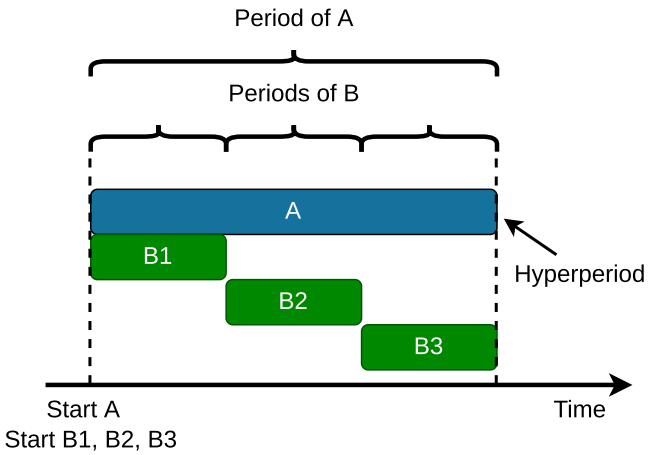


FIGURE 3. The period of stream A is three times the period of stream B. For modelling purposes, the hyperperiod is introduced, i.e., all streams are assumed to have that larger period. To cover the full duration of the hyperperiod, B is modelled by three consecutive copies B1, B2, and B3.

The hyperperiod H of a set of streams $\mathcal{S}$ is the least common multiple of the periods of the streams. Let $s \in \mathcal{S}$ be a stream with period p_s. A schedule for all streams in $\mathcal{S}$ contains $\frac{H}{p_s}$ consecutive replications of s, each having the hyperperiod as period. Scheduling algorithms typically consider transmission times, earliest and latest transmission offsets, and deadlines relative to the beginning of the hyperperiod. Figure 3 depicts an example with two streams A and B. The period of stream A is three times the period of stream B. A schedule for both streams thus contains only one period of stream A and three periods of stream B. A schedule for a set of TT streams in a TSN network consists of the transmission offsets of all streams at their respective talkers, and GCL configurations for all bridges. Transmission offsets of frames at bridges along their path follow implicitly. Schedules must be periodic, i.e., repeatable an indefinite number of times. The hyperperiod of a set of streams is the period of schedules for these streams.

B. SCHEDULING CONSTRAINTS

Given a problem instance for the TSN scheduling problem, i.e., a set of TT streams and a network topology. Every schedule which complies with the real-time requirements of all TT streams is considered a valid solution of the TSN scheduling problem. Such schedules are denoted as valid schedules in the TSN scheduling literature. The following constraints restrict the set of all possible schedules to the set of valid schedules.

1) BRIDGE DESIGN

TSN bridges are currently assumed to be store-and-forward bridges. Frames cannot be forwarded by a bridge before they have arrived at the egress queue. The duration of a transmission depends on the transmission rate of the sending egress

port, the size of the frame, and the propagation delay of the used link. The processing delay of the bridge must also be considered.

2) EXCLUSIVE LINK USAGE
No two frames can be in transmission over a link in the same direction at the same time.

3) DEADLINES
A stream meets its deadline when all of its frames arrive at the stream's destination before its deadline. For multicast streams, this holds for all destinations.

4) ROUTING
The frames of a stream follow some routing. Every instance of the same stream follows the same routing.

5) FRAME ORDER
There is no reordering of frames of the same stream. Frames that are sent earlier arrive earlier than frames sent later. This holds hop-by-hop and end-to-end. The source node of a stream sends frames of a stream in-order. There are no duplicates, i.e., a frame cannot be replicated by a bridge.

6) FIFO QUEUES
The order of frame arrivals at an egress queue must match the order at which frames are sent.

7) QUEUE SIZE
Frames of scheduled traffic must not be dropped for any reason.

8) GATE CONTROL
If a frame waits in an egress queue, it can only be sent when the respective gate is open. The gate must stay open until transmission finishes.

9) TRANSMISSION SELECTION
If multiple gates of an egress port are open and frames in the respective queues are waiting for transmission, the queue with the highest priority is the next queue to dispatch a frame.

10) ADDITIONAL FEATURES
Various modifications of the problem are presented in the literature. Additional constraints may be needed to model these problems. For example, multiple queues can be reserved for TT traffic per egress port. Queue assignment of streams must be modeled in this case. We discuss these problem modifications in Section III-E.

C. FINDING A SCHEDULE VS. OPTIMIZATION
There may be multiple schedules for a given problem instance. In fact, most problem instances have a large number of schedules as possible solutions. So far, the definition of the scheduling problem does not differentiate between these solutions. A common way to compare solutions is to introduce an objective function. Such a function maps solutions to real numbers. The solution to an optimization problem is the schedule which minimizes or maximizes the objective function, i.e., has a smaller or larger objective value than any other schedule. Examples for objectives are minimizing end-to-end delays or jitter of TT streams. Another possible objective is minimizing the flowspan, i.e., the duration such that all frames have arrived at their respective destinations.

D. COMPUTATIONAL COMPLEXITY
The problem of deciding whether there is a valid schedule for a set of TT streams in a TSN network is known to be NP-complete [7] in general as Bin Packing can be reduced to it. This even holds without queuing [28]. NP is a class of decision problems, i.e., contains only problems which can be answered by either *yes* or *no*. Finding a schedule or finding an optimal schedule are not decision problems. Therefore, they are not contained in NP. However, they are computationally at least as hard as the question whether there is a schedule.

E. PROBLEM EXTENSIONS AND RESTRICTIONS
The definition of the basic problem in Section III-B only describes the common properties of the problems in the literature reviewed in this survey. Much research work focuses on special cases or problem extensions with additional constraints. This section introduces these problem variations in a general way such that they are clear in the remainder of this survey.

1) JOINT ROUTING
The definition of Section III-B assumes that the routing of every stream is a predefined part of the input and fixed. Much research work is dedicated to a variation of the scheduling problem with joint routing, which relaxes this assumption. In contrast to the basic problem, the routing of streams is variable and computed simultaneously with the schedule. This gives the scheduling algorithm more flexibility, as streams can be routed to omit heavily loaded links and thus conflicting scheduling constraints. A common approach is that the algorithm gets a set of possible paths as input for every stream, and it selects one per stream as the stream's routing. Other algorithms select arbitrary paths. Both approaches are possible due to IEEE 802.1Qca [15] as the standard allows arbitrary paths to be configured for every stream.

2) RELIABILITY
Research work dedicated to joint routing and scheduling can take reliability considerations into account. Such works define a model of possible faults and their probabilities. Scheduling algorithms can compute schedules which meet the real-time requirements of all streams with high probability for a given fault model. These schedules are denoted as robust schedules relative to a given fault model. For instance, scheduling approaches can compute schedules which are

robust against single link failures. This can be achieved by introducing redundant streams with the same payload and routing them through disjoint paths.

3) GCL SYNTHESIS

GCLs for all egress ports must be constructed. One possible approach is to open the gates for scheduled traffic at the beginning of a hyperperiod and never closing them. However, this approach comes with the drawback that no other queue can send. This may be necessary to protect other TT streams with tighter bounds by avoiding congestion in the queues for TT traffic.

Another common approach is to use a postprocessing scheme after scheduling transmission offsets, e.g., in [28] and [29]. GCLs are constructed such that the gate of a queue is opened when a transmission from this queue should start according to the schedule. The respective gate is closed when the transmission is finished according to the schedule. This approach allows a scheduler to use gates to delay frames. However, the number of available GCL entries is limited in real hardware bridges. Therefore, scheduling transmission offsets and synthesizing GCLs can also be considered in a joint scheduling algorithm instead of a postprocessing, e.g., in [30] and [31].

4) QUEUING

Queuing can cause serious problems for schedules of streams with real-time requirements [29]. Frames can get lost in non-deterministic events, such as link or end station failure. A frame missing in an egress queue may result in another frame being dispatched earlier than expected and scheduled. As a result, this frame may change the arrival order in some egress queue, ultimately resulting in a stream missing its deadline. Such problems can be avoided in two ways. First, by avoiding queuing at all. Second, by not allowing frames to wait in the same egress queue at the same time. In this way, it is not possible that some frame is dispatched earlier than scheduled due to a missing frame in an egress queue. These restrictions are not imposed by bridges according to IEEE 802.1Q [14]. Instead, they are considered during scheduling such that a scheduling algorithm only computes schedules robust against these non-deterministic events. In the following, we discuss problem extensions and restrictions from the literature.

a: UNRESTRICTED QUEUING

Allowing frames of different streams to be in the same queue at the same time is denoted as unrestricted queuing. Figure 4(a) depicts a schedule by showing frame arrivals and transmissions of a single bridge. The schedule shows two streams, A and B, with two frames per period. The queuing state is shown implicitly. A frame is queued at the same time with all other frames that arrive before the frame is transmitted. Thus, the frames A1 and B1 are in the egress queue at the same time. If A1 does not arrive according to the schedule, e.g., due to a permanent link failure, B1 is transmitted earlier than scheduled. This is the case in the second period depicted in Figure 4(b). Consequently, B1 arrives earlier than scheduled in some other egress queue. This may result in some other frame experiencing more queuing delay than scheduled, ultimately leading to a missed deadline.

b: ISOLATION

The problems of queuing in case of non-deterministic events can be solved by not allowing frames of different streams to be in the same queue at the same time. If a frame is missing in a queue and no other frame is scheduled to be queued at the same time, no other frame can be transmitted earlier than scheduled. This approach is denoted as *frame isolation* in the literature. It was introduced in [29]. Figure 4(c) shows a schedule valid with frame isolation. If A1 does not arrive at the bridge, the schedule of B1 and B2 is unaffected.

c: NO-WAIT SCHEDULING

In no-wait scheduling, frames are dispatched and sent immediately after arriving at an egress queue. Queuing is not allowed. The rational of this constraint is to avoid all consequences of non-deterministic events related to queuing. It was introduced to the domain of TSN in [28]. Figure 4(d) depicts a no-wait schedule for two streams. All frames are sent immediately after arrival. For example, B1 is received and transmitted after A1 and before A2 in the same period.

d: QUEUE ASSIGNMENT

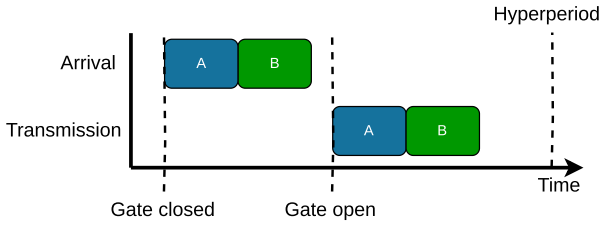
Instead of restricting queue usage, the scheduling problem can also be extended by allowing more than one queue per egress port for TT streams [29], [31]. A scheduling algorithm for such a problem not only schedules transmission offsets and GCL entries, but also assignments of TT streams to egress queues. This is especially interesting with respect to frame isolation, as frames of multiple streams can simultaneously wait for transmission by the same egress port in different queues.

5) INTEGRATION OF AUDIO VIDEO BRIDGING

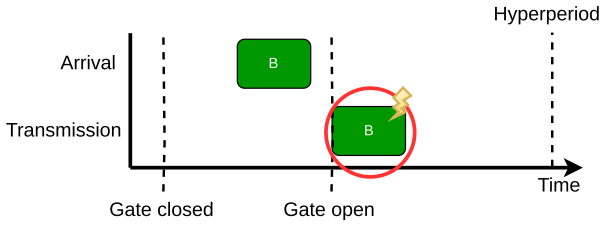
TT streams and AVB traffic can coexist in the same network at the same time. TSN bridges may support to use the CBS and the TAS in parallel according to [6]. TT streams and AVB streams compete for the same links, but use different queues in the egress ports. Therefore, the scheduling problem can be extended to also include a set of AVB streams as input. They are scheduled at their respective talkers, and considerations for the behavior of the CBS must be included during scheduling.

6) INTEGRATION OF BE TRAFFIC

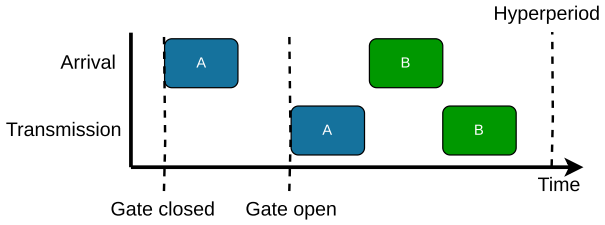
BE streams have no real-time requirements, they are generally aperiodic and unknown a priori such that they cannot be scheduled. However, some schedules may be beneficial for BE traffic. For instance, large bursts of TT traffic within a long hyperperiod could be avoided to facilitate frequent



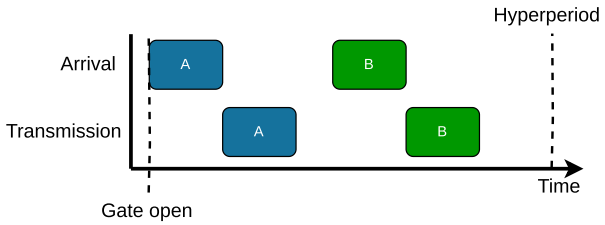
(a) Unrestricted queuing without fault. Frames A and B wait in the egress queue at the same time.



(b) Unrestricted queuing with fault. Frame B is sent earlier than scheduled if frame A was lost.



(c) Frame isolation. Frame A waits at a closed gate for some time before transmission.



(d) No-wait scheduling. Frames are transmitted immediately after arrival.

FIGURE 4. Queuing restrictions from TSN scheduling literature. Frame arrivals at an ingress port and transmissions at an egress port of the same bridge are shown. Processing delays are omitted to increase comprehensibility.

transmission opportunities for BE traffic, which may reduce the delay of BE traffic. Another example is avoiding GCL entries unless they save substantial capacity for other traffic. For each GCL entry, a guard band is needed within which frame transmissions cannot start. Therefore, compact schedules maximize capacity for BE traffic [28], [32].

7) DYNAMIC RECONFIGURATION

An entirely different problem related to the basic problem is dynamic reconfiguration of existing schedules. Such reconfigurations are necessary when streams are removed or new streams should be integrated into a schedule. While removing streams is rather easy, adding new streams to an existing schedule can be complicated for two reasons. First, the transmission offsets of already scheduled streams may have to be changed. Second, links and egress queues are occupied by earlier scheduled streams, which places constraints on possible transmission offsets for new streams. The runtime of a scheduler during reconfiguration must be very low in many scenarios, such as in automotive use cases [33]. This is due to fast changing real-time requirements and traffic patterns of safety-critical applications. Recomputing the whole schedule with offline algorithms may be computationally infeasible in such cases.

8) MULTICAST

Streams in bridged Ethernet networks can be multicast streams according to [14]. Multicast streams have more than one listener as destinations. Therefore, the routing of a multicast stream is a tree. A multicast stream can be modelled by a set of unicast streams. However, only a single copy of a frame is transmitted per hop in TSN. Thus, this modelling is not appropriate. Scheduling algorithms may contain considerations for multicast streams instead of assuming all streams to be unicast. Joint routing approaches must compute trees instead of paths for every stream.

9) TASK SCHEDULING

Tasks are applications running on end stations. They are executed periodically. Their execution depends on data received via TT streams. Additionally, they can send TT streams after they processed some received data. Scheduling algorithms for TSN can schedule tasks and TT streams in a joint approach.

F. OPTIMIZATION METHODS

We classify the scheduling algorithms in the literature in exact and heuristic approaches. Exact approaches compute a schedule, or an optimal schedule if an objective is given, if one exists, or prove the problem instance infeasible. Heuristic approaches do not guarantee to find an optimal schedule. Instead, they try to find reasonably good solutions within short time. In the common case, they cannot deduce whether a problem instance is infeasible, nor is finding a solution guaranteed if one exists. In this section, we introduce common solution techniques and explain their basics.

1) EXACT APPROACHES

As the Scheduling Problem for TSN is NP-complete, there is probably no polynomial-time algorithm to compute TSN schedules. Therefore, it is reasonable to rely on the advances of the past decades in mathematical and combinatorial optimization. All exact solution approaches in the literature are based on the following four techniques.

a: INTEGER LINEAR PROGRAMMING

An Integer Linear Program (ILP) describes the space of possible solutions to a problem with linear inequalities. Every assignment of variables which fulfills all inequalities

$$\begin{aligned} \min\ & 2x + 3y - z \\ \text{s.t.}\ & 4x + 3y \leq 7 \\ & x - 2z \leq 4 \\ & 3y + 2z \leq 10 \end{aligned}$$

FIGURE 5. Example of an ILP model.

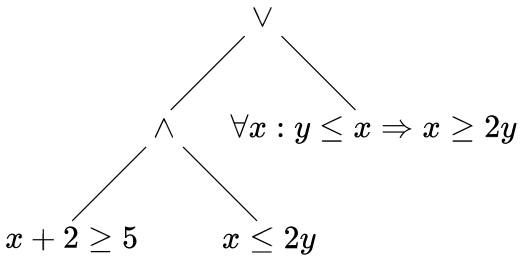


FIGURE 6. Example of a formula used in SMT solving. The basic structure is a formula from propositional logic, but predicates from other theories may be used as atomic formulas instead of Boolean variables.

corresponds to a solution of the problem and vice versa. Some variables may be restricted to take only integer values. A linear objective function may describe the quality of solutions. Figure 5 depicts an ILP which minimizes an objective function. ILP solvers compute a feasible assignment which minimizes the objective function. Every encountered solution in the solution process corresponds to an upper bound on the objective value of the optimal solution. Additionally, the solver can infer lower bounds for the objective value of the optimal solution during the solution process. So even when finding the optimal solution is not possible in reasonable time, ILP solvers yield estimations of the maximum gap to the optimum. Widely used state-of-the-art ILP solvers are CPLEX [34] and Gurobi [35].

b: SATISFIABILITY MODULO THEORIES

Satisfiability Modulo Theories (SMT) solvers find solutions to problems described by first-order formulas. Formulas model a problem with variables and predicates which are connected by logical operators. Besides Boolean variables, SMT solvers allow formulating predicates in other logical theories and use them as atomic formulas. SMT solvers have an interface for theory-specific solvers so that the problem can be modelled with the best suitable theory. Examples of theories are the theory of linear arithmetic with integers or the theory of bit vectors. Figure 6 depicts a formula with predicates from the theory of linear arithmetic with integers. The basic structure of a model is a formula from propositional logic, but predicates from integer arithmetic are used as atomic formulas.

The solver searches for an assignment of the variables that evaluate the formula to *true*. It uses techniques from SAT solving to reason about satisfiability, combined with theory-specific solvers for conjunctions of predicates. SMT solving is only about finding some satisfying solution. When the best assignment regarding some objective function is computed, the term OMT is used. Z3 [36] is a widely used SMT solver which can also be used for optimization.

c: CONSTRAINT PROGRAMMING

Constraint Programming (CP) is a general solution approach to combinatorial problems. The set of feasible solutions to a problem is described in a declarative way. In this sense, ILP and SMT solving are special cases of CP solving. However, CP solvers use backtracking, local search, and constraint propagation techniques to solve CP models as opposed to ILP solvers. Another relevant case of CP is the restriction of variable domains to a finite set. CP-SAT is a widely used CP solver [37].

d: PSEUDO-BOOLEAN OPTIMIZATION

Similar to ILPs the solution space of a problem in Pseudo-Boolean Optimization (PBO) is modelled with linear inequalities, but all variables must be binary. However, instead of using mathematical optimization as in ILP solving, techniques from SAT solving like propagation and conflict refinement are employed. A linear objective function can be minimized by adding it as a constraint to the model with some bound. The solver is called multiple times with different bounded objective constraints. Every infeasible solver run gives a lower bound on the optimal objective values. Every solution yields an upper bound on the optimal solution. The optimal solution is found, with respect to some minimal precision, when the gap between lower and upper bound is smaller than the minimal precision provided by the user.

2) HEURISTIC APPROACHES

Because finding optimal solutions for realistic problem instances is infeasible in many cases, heuristic algorithms are used. Such algorithms are used to find suitable solutions in reasonable time, generally without knowing whether there are better solutions. Metaheuristic approaches are common algorithms that can be applied for a wide range of problems. Alternatively, there are heuristics that use problem-specific knowledge for many problems, and there may be combinations of both.

a: GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

Greedy Randomized Adaptive Search Procedure (GRASP) is a metaheuristic which can be adapted to various problems. Its building blocks are a greedy-randomized algorithm to construct initial feasible solutions, and a local search algorithm. The greedy randomized algorithm incrementally constructs a solution by making random decisions among the set of decisions with the smallest increase in cost until a feasible solution is found. The local search explores neighboring solutions, i.e., solutions with minimal changes, to the intermediate

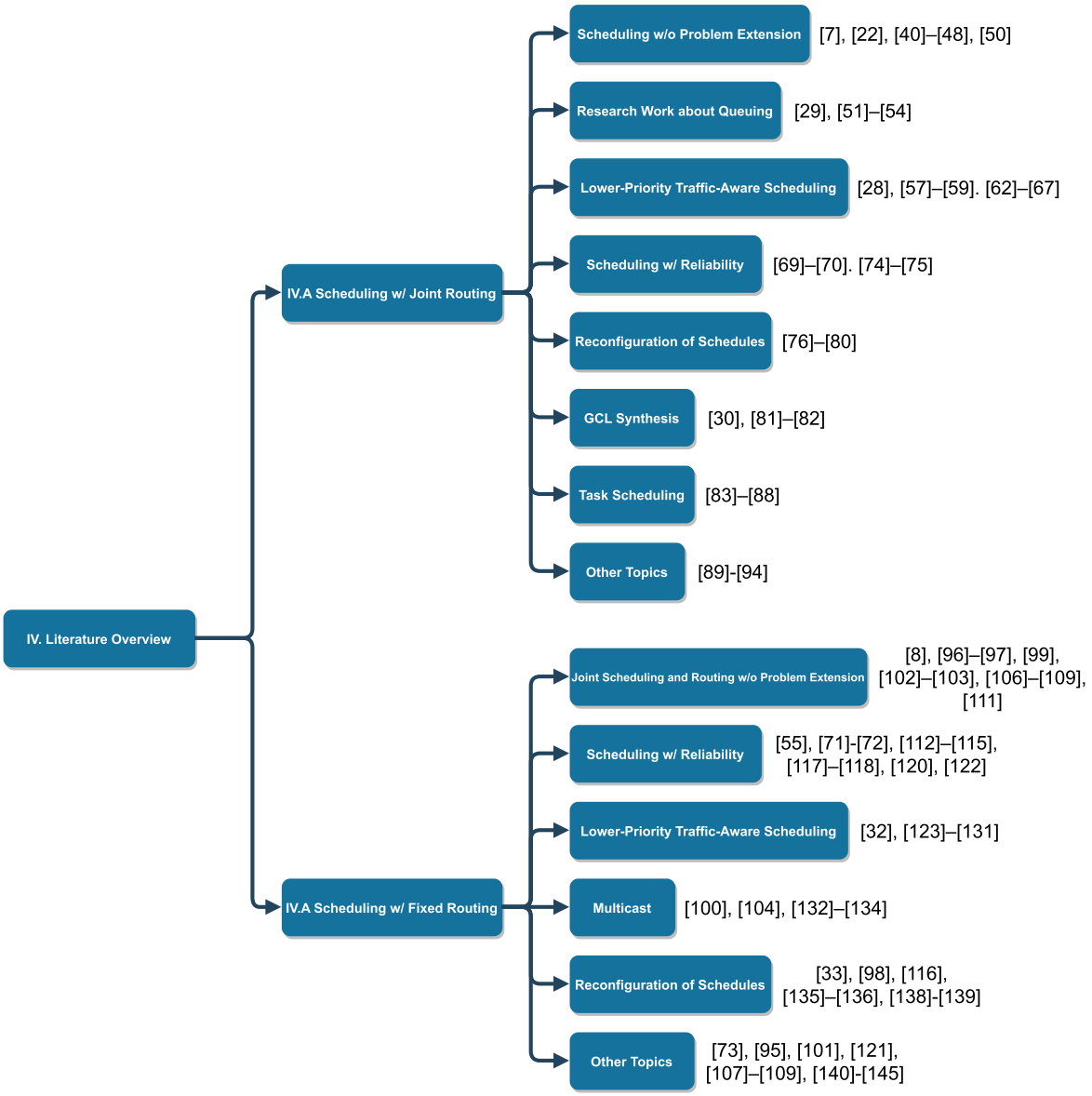


FIGURE 7. Classification of the surveyed research works in the scope of TSN scheduling.

solution. It explores the solution space until it finds a local optimum. Both steps are repeated a predefined number of times and the best encountered solution is returned. To adapt GRASP for a specific problem, a greedy randomized algorithm to generate initial solutions and a local search algorithm must be constructed.

b: TABU SEARCH

Tabu Search is a metaheuristic to systematically explore the solution space. It uses an initial solution as start and moves to the best neighboring solution. The algorithm keeps a tabu list of previously visited solutions or changes to solutions to avoid walking cycles in the solution space. Only neighboring solutions or changes to solutions which are not contained in the tabu list are considered for the move. The best encountered solution after a specific number of moves is returned. To construct a problem-specific heuristic, a heuristic to generate an initial solution and a function returning the possible changes to some given solution must be built.

c: SIMULATED ANNEALING

Simulated Annealing (SA) is a metaheuristic used to find good approximations of the global optimum of an optimization problem. It is inspired by cooling processes in physics. A global variable for temperature is used. Temperature decreases slowly to 0 in discrete steps. In each step, a neighboring solution is randomly selected, and the objective function is evaluated. The probability of moving to a neighboring solution depends on the current temperature and the objective value of the considered solution. A move to a neighboring solution which is worse than the current solution is possible with small probability to escape from local optima.

As temperature decreases, the probability of moving to solutions with worse objective value vanishes. The best solution encountered after some acceptance criterion holds is returned. To adapt SA to a specific problem, a heuristic to generate an initial solution and a function returning the possible changes to some given solution must be built. Additionally, the way the temperature is decreased and the acceptance criterion must be selected.

d: GENETIC ALGORITHMS

Genetic algorithms (GA) are a metaheuristic approach inspired by evolution processes and natural selection in biology. Candidate solutions are considered as individuals. Chromosomes represent properties of these individuals and are coded into bitstrings. At every point in time, there is a pool of individuals, i.e., the population. New individuals are constructed from two or more existing solutions, i.e., genetic crossover is performed. Individuals may be altered randomly, i.e., their chromosomes are mutated. When transiting to the next generation, some individuals die and are removed from the population. The probability of dying for an individual depends on its fitness. The fitness function is the optimization objective of the modeled problem. The best individual encountered after some number of generations is returned. As in biology, high-quality solutions have a higher probability to survive and reproduce, which in terms yields new high-quality solutions. To construct a problem-specific heuristic, a heuristic to generate initial solutions must be constructed. Suitable crossover as well as mutation and selection mechanisms have to be used. Parameters like population size, stopping criterion, and probabilities for selection and mutation must be designed.

e: LIST SCHEDULING

List scheduling (LS) is a metaheuristic to schedule tasks on identical machines. The tasks are sorted in a list according to some measure of priority. In every step, the first task in the list is selected. If a suitable machine is available, the task is executed on this machine, otherwise the next task in the list is selected. These steps are repeated until all tasks are executed. Considering streams as tasks and end stations as machines yields a heuristic for TSN scheduling. A well-known heuristic from the scheduling literature can be considered to be special case of list scheduling. As-soon-as-possible (ASAP) scheduling orders streams by priority and schedules them one by one at the earliest possible time along their paths.

f: MACHINE LEARNING

Machine learning is the generic term for a wide range of methods. Tools from linear algebra, statistics, and probability theory are used to construct mathematical models that can make decisions or construct solutions to a problem. The construction of such a model is denoted as *learning* or *training*. Typically, it takes a large amount of time and computational effort to train a model, but answers to request can be obtained really fast afterwards. Examples of machine learning methods are deep learning and reinforcement learning. However, the details of these methods are way beyond the scope of this survey. We refer to [38] and [39] for an introduction.

IV. LITERATURE SURVEY

In the following section, we give an overview of the literature about TSN scheduling. We categorize research work based on whether scheduling with fixed routing or joint routing is considered. Both sections are further grouped by the main topics of the respective papers. Comparability of techniques and results of research works in the same group is ensured by this classification. Figure 7 depicts this classification.

A. SCHEDULING W/FIXED ROUTING

We give an overview of research works which only deal with the scheduling of TT streams. In all papers presented in this section, the routing of TT streams is fixed and given as input to the scheduling algorithm. Such scheduling algorithms cannot change the routing during scheduling in case of conflicting streams. We group publications in categories based on similar topics, like model assumptions or problem extensions.

1) SCHEDULING W/O PROBLEM EXTENSIONS

We discuss publications solving the unmodified scheduling problem.

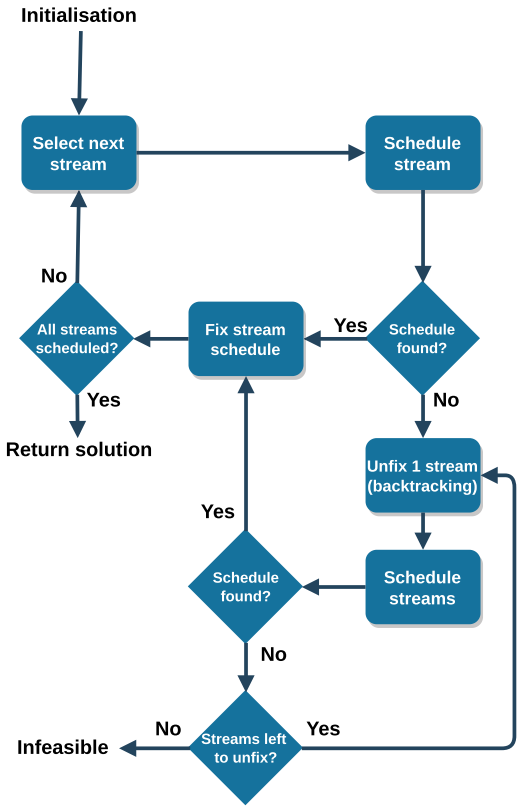


FIGURE 8. Incremental approach of Steiner [7]. Similar ideas were used by other approaches, e.g., in [29].

Early work about scheduling of TT traffic in Ethernet networks was conducted by Steiner [7]. Even though this

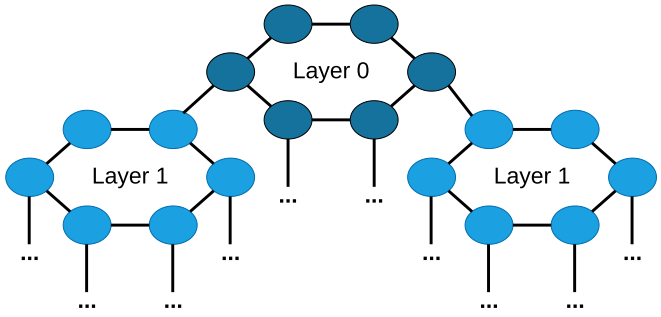


FIGURE 9. Multi-layered ring topology used in [40].

work is not specific to TSN, it influenced many research works covered by this survey. The author proposes the use of SMT solving for the scheduling of TT streams. An incremental approach is presented. Figure 8 depicts this approach. Streams are scheduled one after another. Schedules of already scheduled streams are fixed in later iterations. Backtracking is used in case of infeasibility, i.e., the schedule of some stream is unfixed and the stream is scheduled again simultaneously with the new stream. Backtracking is repeated until a schedule is found, or no stream schedules are left to unfix. This idea was adopted by many later works for TSN scheduling, e.g., [29] and [31].

Oliver et al. [31] give an SMT model based on mapping streams to transmission time windows of egress queues. The number of these transmission windows is fixed per egress port, and their placement and size is computed by the scheduling algorithm. As a side effect of using a fixed number of transmission windows, the number of gate events and thus guard bands is limited, even though the authors do not explore this matter. The authors use isolation to restrict the problems imposed non-deterministic behavior, e.g., frame loss. Two queues per egress port are dedicated for TT traffic. They evaluate the solving time of their approach with respect to the number of streams and the number of transmission windows per egress port. Their results indicate that the solving time increases exponentially with the number of streams. However, for reasonable numbers of streams and transmission windows, solving time is more sensible to the number of transmission windows. A comparison to the SMT from [29] shows that the window-based approach with one window per egress port is faster in finding a schedule. The average jitter is significantly reduced when the number of transmission windows per egress port is increased.

Steiner et al. [41] suggest the SMT model from [31] as a starting point for the standardization of TSN scheduling mechanisms. They demonstrate their model by reporting the same evaluation results as in [31] with a reduced number of transmission windows.

Hellmanns et al. [40] extend the Tabu Search algorithm of [28] for no-wait scheduling. They construct a 2-stage approach for hierarchical networks which consist of multiple rings on different layers. They argue that such topologies are common in factory automation. Figure 9 depicts a model of such a topology. First, they schedule streams with talker and listener in the same ring. This step is done individually for every ring. No queuing is allowed in these schedules. Then, they simulate the transmission of all streams with end points in different rings as if they were sent at the same time from their respective talkers. No queuing restrictions apply to these streams, i.e., unrestricted queuing from Section III-E4.a is allowed. If all streams meet their deadlines in this simulation, the simulated behavior is used as schedule. They compare this approach with scheduling all streams at once with the Tabu Seach algorithm. Their evaluations demonstrate that the proposed 2-stage scheduling scales better for problem instances with many streams compared to the original Tabu Search approach. The latter does not produce results for more than 1000 streams due to memory limitations. The 2-stage scheduling is two orders of magnitude faster in the special case of multi-layered ring topologies. The authors report that the number of needed GCL entries is significantly reduced by the 2-stage approach.

Another heuristic for no-wait scheduling is proposed by Zhang et al. [22]. They analyze how frame transmissions may conflict and derive the range of possible transmission offsets per frame. A comparison to the SMT of [31] and the ILP of [28] shows clear performance benefits of the proposed heuristic. The SMT was able to schedule about 300 streams, while the ILP scheduled about 1000 streams, and the heuristic scheduled 1200 streams in the evaluation scenario.

Kim et al. [42] give a heuristic algorithm to compute valid schedules, and a post-processing to reduce end-to-end delays. Streams are ordered by priority and are scheduled one after another. The individual frames of a stream are scheduled along the stream's path. The hyperperiod is divided into intervals and every frame is assigned to the earliest unoccupied interval. The presented evaluations indicate that end-to-end delays are reduced by up to one third per stream in the evaluation scenarios.

The authors of Kim et al. [43], [44] propose a genetic algorithm to schedule TT traffic in automotive scenarios. Genes encode the scheduling order of frames. Frames are scheduled as soon as possible according to this order and along the respective stream's path. The objective function used to compare scheduling orders is the weighted sum of end-to-end delays, jitter, and bandwidth utilization of the corresponding schedule. As in [28], a schedule compression algorithm is employed to reduce the bandwidth occupation of guard bands. The proposed approach outperformed random schedules regarding all three metrics in almost all evaluation scenarios. The approach from [42] is also outperformed with regard to the used objective.

Ansah et al. [45] present a scheduling algorithm in the special case of a line topology where all talkers converge in a single bridge. Based on this method, they also give an algorithm to compute GCLs in such a topology if the streams are schedulable.

The special case of an in-vehicle network with only a single hop is analyzed in [46]. They assume all traffic streams to be send continuously and belonging to different traffic classes and egress queues. Essentially, they implement round robin traffic shaping with the TAS. However, we remark that real in-vehicle networks are more complex and thus the gained insights are limited.

The authors of [47] compare the suitability of a large set of metaheuristics for TAS scheduling. They maximize the number of scheduled streams for the same problem instance with various functions of a metaheuristic library. The authors observed the best results with math based and system based heuristics and interpret this as a hint for future research directions.

Vlk et al. [48] propose a heuristic algorithm to schedule very large-scale problem instances. The algorithm shares many similarities with the well known DPLL algorithm from SAT solving [49], e.g., probing, backtracking in the case of conflicts, and restarts. Frames are scheduled one by one. If a conflict arises, all decisions are reverted up to the conflicting frame. The authors compare the heuristic with SMT-, ILP-, and GRASP-based algorithms. The schedulability of an approach with respect to some set of problem instances is the fraction of solvable problem instances within some time limit. The proposed heuristic outperforms all other approaches regarding schedulability and solving time. In fact, they were able to schedule instances with up to 10812 streams in a tree-like topology with 2000 nodes. This result outperforms all other approaches in the literature. Evaluations with a real-world instance from avionics are also presented.

Wang et al. propose a deep reinforcement learning approach for no-wait scheduling in [50]. They train machine learning models for various network topologies. The model aims to reduce the maximum arrival time among all frames to reduce the number of guard bands. For networks with up to 9 bridges and 10 end stations, the authors report solving times of at most 400 s.

2) RESEARCH WORK ABOUT QUEUING

We highlight works which allow or deal with the implications of queuing.

Craciunas et al. [29] construct an incremental SMT model to schedule TT streams based on [7]. They define flow isolation and frame isolation as properties of a schedule to prevent some sources of non-determinism, e.g., single link failures. They present models to compute schedules with either flow or frame isolation. Besides isolation in the time domain, they also employ isolation in the spatial domain by the possibility of assigning different streams to different queues. The authors identified the problem of clock synchronization errors and introduce gaps between frame transmission to cope with this problem. They compare the effect of frame and flow isolation to the solving time of their SMT model. Their evaluations indicate that flow isolation reduces solving times compared to frame isolation. However, more problem instances can be scheduled with frame isolation.

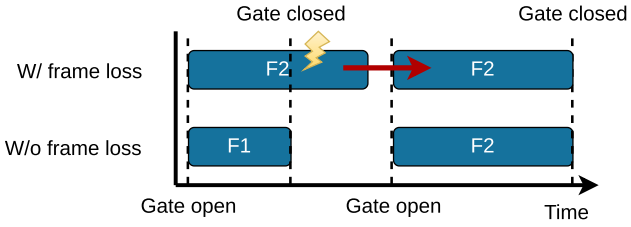


FIGURE 10. Size based isolation proposed in [53]. Frame transmissions from an egress port connected to an end station are shown. Assume F1 and F2 are scheduled to wait some time in the same egress queue. If the first frame F1 has not arrived at the egress queue, F2 cannot be transmitted in the time slice dedicated for F1 as it is too short.

Vlk et al. [51] investigate the effect of the isolation constraints from [29] on schedulability. When a frame is lost during transmission and does not reach the next egress queue as scheduled, another frame may be dispatched earlier than scheduled from this queue. This frame in term can cause more non-determinism on its path. Not allowing frames of different streams to be in the same queue at the same time solves this problem, but reduces the solution space considerably. A modification for bridges implementing the TAS is proposed to cope with this conflict. Queues with this modification check whether the next frame is the correct one with respect to the schedule. If this is not the case, the queue idles until the next frame transmission is scheduled. A comparison shows clear benefits regarding schedulability. The number of streams which are scheduled to arrive before their deadline is also significantly increased compared to isolation models.

The authors of [52] present a heuristic to schedule streams with queuing. The heuristic is based on transmission windows similar to [31]. In contrast to earlier works which include queuing in their model [29], [31], they drop isolation constraints. Network calculus is employed for a worst-case end-to-end latency analysis. They minimize the occupation percentage of egress ports, i.e., the percentage of the hyperperiod which is reserved for TT traffic. In this way, long and frequent time intervals for lower-priority traffic are scheduled. Their evaluations indicate that their approach is superior regarding end-to-end delay and schedulability of streams compared to earlier works from the same authors.

Chaine et al. [53] use queuing for jitter control. They propose to schedule streams without queuing at all egress ports except for egress ports connected to end stations. Frames are buffered in these egress ports and are released such that jitter constraints are satisfied. The authors present a novel isolation approach, denoted as *size based isolation*. Frames must be buffered in increasing frame size order if they are stored in the same queue. Two GCL entries are used to close and open the corresponding gate between two frame transmissions. In this way, frames cannot be transmitted during an earlier time slice than scheduled if another frame is missing in the queue, as earlier time slices are too short. Figure 10 depicts such a scenario. The authors give an ILP model to

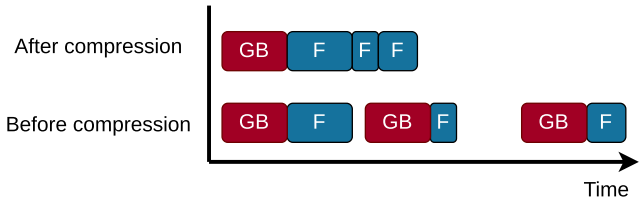


FIGURE 11. Effect of the schedule compression algorithm from [28]. Scheduled frame transmissions (F) and guard bands (GB) over a single link are shown.

compute schedules with their approach. A comparison of their approach to an unspecified approach for latency minimization demonstrates that their approach reduces scheduling time significantly. However, this comes with the cost of higher latencies.

Bujosa et al. [54] propose a heuristic scheduling algorithm which handles queue assignment of streams. Instead of scheduling frames or streams one after another on their entire path, they schedule all transmissions over a single link before scheduling the transmissions over another link. They present results about the scalability and schedulability of their approach compared to a CP approach from the literature [55]. Not surprisingly, scheduling is significantly slower with a CP approach compared to a heuristic.

3) SCHEDULING W/OTHER TRAFFIC

The schedule of TT streams may affect other traffic classes. AVB and BE traffic cannot be scheduled, but QoS metrics of these classes can be influenced when they are taken into account during the scheduling of TT streams. We summarize works with such considerations.

Dürr et al. [28] present an ILP and a Tabu Search algorithm to compute no-wait schedules for TT streams. They model the problem with job-shop scheduling, a widely used modelling framework in the scheduling literature. The authors measure the solving times of their Tabu Search and conclude that the network topology and size have no influence on solving times. They minimize the flowspan to construct a large time slice for BE traffic at the end of the computed schedules. They propose a compression algorithm as post-processing for schedules which aims to reduce the number of GCL entries needed to deploy a schedule. The authors note that this increases the available bandwidth for BE traffic as the number of guard bands is reduced. Figure 11 depicts the effect of the schedule compression algorithm. They report that the number of GCL entries can be reduced by 24% on average. Parts of the content of this work are also featured in the PhD thesis of Nayak [56].

The authors of [57] give an ILP to compute schedules for TT streams and additionally present a GRASP heuristic to schedule AVB streams. They restrict queuing by enforcing frame isolation. Their heuristic computes a routing for AVB streams such that they meet their deadlines. It reduces the search space by only considering a fixed number of shortest paths for every pair of nodes as possible routings. The schedule of TT streams, computed by their ILP model, serves as input for the heuristic and cannot be changed. They compare their AVB routing with the naïve approach of always selecting the shortest path. The comparison demonstrates that more AVB streams can be scheduled with their approach. A comparison of solving times of their ILP to the SMT from [29] is conducted. They state that their proposed ILP does not scale well for industrial-size instances and further efforts to create a suitable heuristic are needed.

The authors of [58] propose an SDN-based method for traffic bandwidth allocation in safety-critical environments. While this work does not present a scheduling algorithm for the TAS, it gives a method to configure the CBS such that latency requirements of streams are met. They use a particle swarm optimization heuristic for this purpose. Evaluation results for the schedulability of stream reservation messages under varying network utilization by TT traffic are reported.

Santos et al. [59] present an extensive SMT-based modelling of the scheduling problem with openly accessible implementation. Their model contains a range of features known from previous works, e.g., transmission windows, multicast, guard bands, and bandwidth considerations for BE traffic which were not covered by a single approach in the past. The starvation of BE traffic is prevented by restricting a user-defined fraction of a hyperperiod exclusively to be used by other traffic which is related to the approach of minimizing the flowspan [28]. Additionally, unrestricted queuing is integrated which is uncommon in exact approaches so far. The authors mention the limitation of only one gate opening per queue per hyperperiod in the presented model which reduces the available bandwidth for other traffic classes. They evaluate their approach on a realistic sized network and report successful scheduling for up to 10 multicast streams. The model is also used in the well known simulation framework OMNeT++ [60]. The thesis of Santos [61] explains the model in detail.

Houtan et al. [62] compare schedules computed with various objectives for the same SMT model with respect to the QoS of BE traffic. They propose minimization and maximization of frame offsets, with the goal of increases the QoS by grouping frames together. Additionally, they also suggest two objectives which maximize the gaps between consecutive frame transmissions over a link. They integrate frame and flow isolation in their SMT model. Unfortunately, their work lacks a description which one was used in the evaluations. A comparison of the different objective functions indicates that larger gaps between frame transmissions of TT streams increase the QoS of BE streams. For instance, BE traffic may experience less starvation and average latencies are reduced. Figure 12 depicts how BE traffic may benefit from maximizing the gaps between TT frame transmissions. However, we note that the used system model features deadlines for BE traffic, and they measure the number of deadline misses, so a comparison with the other mentioned research works is

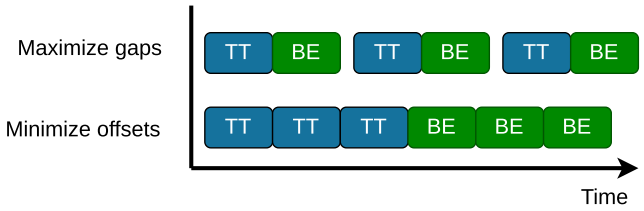


FIGURE 12. Effects of different objective functions to BE traffic in [62]. Frame transmissions of TT and BE traffic over a single link are shown.

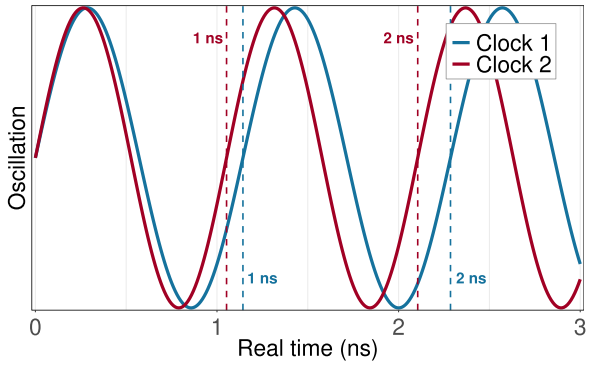


FIGURE 13. Timing signals of two clock oscillators. Clock 1 runs slower than clock 2. Therefore, the difference between both clocks increases over time.

not possible. The solving time for their SMT depends heavily on the objective function used.

The model of [29] is used for an in-vehicle scenario in [63]. The authors present simulation results with typical traffic patterns in such a scenario. They compare end-to-end delays for schedules using the TAS to schedules using the strict priority mechanism. Their results indicate that scheduling with the TAS can ensure real-time requirements of TT streams while the performance of lower-priority traffic is less affected compared to the strict priority mechanism.

Barzegaran et al. [64] give a CP approach to compute transmission windows for TT streams. In contrast to other window-based approaches [31], [41], they assume that not all end stations support TSN. They use a worst-case delay analysis to eliminate solutions that may violate the real-time requirements of the given problem instance. They compare their approach to the algorithms presented in [29], [31], [52], and [57]. They outperform these approaches in terms of solving time, but end-to-end delays and bandwidth utilization are significantly worse compared to [29] and [31]. They also perform simulation runs of their schedules with OMNeT++. The results indicate that their worst-case analysis for end-to-end delays holds but overestimates the simulated delays considerably.

The coexiestence of the TAS and Cyclic Queuing and Forwarding (CQF) in TSN is investigated by Pei et al. [65]. They propose to use CQF for rate constrained traffic with deadlines, i.e., some egress queues per egress port are shaped by CQF. Streams of scheduled traffic and rate constrained traffic are scheduled simultaneously. They are scheduled one after another in least laxity first order, i.e., the next scheduled stream is the stream which deadline expires next. The same approach is used only for scheduled traffic streams as an alternative for comparison. The evaluation shows that the joint handling of scheduled traffic and rate constrained streams results in higher schedulability.

Another approach which combines the TAS and CQF is presented by [66]. They consider the scenario of multiple traffic classes with different real-time constraints. Besides of scheduled traffic with low latency and jitter requirements, there are also two other traffic classes with uncritical periodic streams and best effort traffic. The uncritical periodic streams are assigned by egress queues shaped by the CQF. The authors present a heuristic to compute a schedule for all traffic classes simultaneously. The evaluations show that sorting streams in earliest deadline first order before scheduling is beneficial for the schedulability. In contrast to that, sorting streams by frame size or period reduces schedulability considerably.

Wang et al. [67] propose a combined scheduling scheme for TT and AVB streams. AVB streams are shaped with CQF. They use guard bands to protect TT frames from AVB frames. Their heuristic tries to schedule as many AVB streams as possible while load balancing the traffic amount between the time slots of the CQF mechanism. The authors report that their approach significantly reduces jitter and solving times compared to another approach for CQF.

Huang et al. [68] propose a recursive scheduling heuristic using backtracking. They use a complex in-vehicle topology to evaluate their approach and also include AVB streams in the evaluation scenario. We highlight that they give detailed stream parameters which is rare for real-world use cases. They also include Frame Replication and Elimination for Reliability [17] to cope with frame loss of safety critical traffic. Additionally, an SMT model is presented and compared to their heuristic. The heuristic outperforms the SMT in regard to schedulability, scalability, and end-to-end latencies by far in the evaluation scenario.

4) SCHEDULING W/RELIABILITY

Reliable transmission of data streams is one of the design goals of TSN. Additionally, to hardware features ensuring reliability, schedules can be assembled to mitigate the effects of various faults. We discuss publications which take such considerations into account.

The clock frequencies of two clocks are not exactly equal for technical reasons. This results in so-called *clock drift*, i.e., clocks running with different speeds. Figure 13 depicts this problem. Craciunas et al. [69] extend their model from [29] to cope with clock drift during scheduling. They introduce a parameter for the maximum allowed clock drift into all equations which contain transmission offsets or reception times of frames. Effectively, they merely increase the gap between frame transmissions which is already contained in the model of [29]. Clocks are resynchronized after some predefined out-of-sync detection timeout. The authors present a design space

study which investigates the relationship between the maximum allowed clock drift, the worst-case clock drift rate, the maximum possible diameter of the synchronization spanning tree, and the out-of-sync detection timeout. Their findings on a number of test networks indicate that shorter out-of-sync detection timeouts are needed for higher clock drift rates. The maximum possible diameter of the synchronization spanning tree is negatively affected by higher clock drift rates. Evaluations for a test case regarding schedulability, end-to-end latency, and solving time are conducted. The results show that end-to-end latency increases for higher allowed clock drifts. Maximizing the allowed clock drift yields a maximal robust schedule for a given problem instance, but solving time increases by an order of magnitude compared to setting a fixed maximum clock drift in advance.

Feng et al. [70] consider the scheduling problem in the presence of frame loss. Instead of scheduling redundant streams over disjoint paths, as in [71], [72], and [73], reliability is achieved by multiple transmissions of a stream over the same path. Thus, the proposed approach is only applicable in the case of spontaneous frame loss and temporary link failures. The research work focuses on choosing an appropriate number of repetitions per stream for a trade-off of reliability and network utilization. In contrast to similar works, considerations for AVB and BE streams are also included in the algorithm, as repeated transmissions of TT streams deplete the available bandwidth and may lead to starvation of other traffic otherwise. The presented algorithm uses the SMT model from [29] as a sub-routine. Their results show that increasing the fault probability leads to a higher number of retransmissions which in turn results in less available bandwidth for BE traffic.

In later works, Feng et al. [74] studied a similar problem, but also considered ACK and NACK messages and queue assignment of streams. In contrast to [70], every TT stream is sent exactly twice. Transmission windows for BE streams are computed after the scheduling of TT streams. The scheduled transmission intervals for the retransmissions can be used to transmit BE traffic when no retransmissions are needed.

Dobrin et al. [75] present a heuristic scheme to schedule streams with reliability considerations. They consider transmission losses for frames such that only one frame is affected by a fault at a time and the fault is fixed by some predefined number of retransmissions. Their approach first tightens the deadlines to take some number of retransmissions into account. Then, they schedule streams in earliest deadline first order. Additional considerations for rate constrained traffic are also included in their scheme, following the scheduling of the TT streams. Unfortunately, no evaluations are presented. The authors note that future works will address more realistic fault models.

5) RECONFIGURATION OF SCHEDULES

The reconfiguration of schedules has two distinct meanings in the context of scheduling for the TAS. First, an update to an existing schedule must be computed when the set of streams or the requirements change. Second, a modified schedule from the first case must be deployed to hardware devices, i.e., GCLs and transmission offsets are reconfigured. This section focuses on the first meaning as the deployment of schedules is out of scope for scheduling algorithms. Adding and removing streams from an existing schedule is necessary in dynamically changing environments, e.g., automotive use cases. While removing a stream is straightforward, adding new streams may require more effort. We summarize research works concerned with this problem extension.

Raagaard et al. [76] propose an algorithm for online scheduling of new TT streams in an existing schedule. They use a heuristic which schedules streams as early as possible such that schedules comply with isolation. When a new stream should be added to an existing schedule, they calculate whether there is a starting offset such that the stream can be scheduled without changing the existing schedule. If this is not possible, the stream is assigned to unused queues of the egress ports along the stream's path. The authors state that adding streams to an empty schedule resembles the worst case of removing all streams from a schedule and adding a set of new streams. Thus, they evaluate how many streams can be scheduled in a specific time. They report that their heuristic is able to schedule about 1300 frames per second in medium-sized test cases.

Pang et al. [77] compute schedules with an ILP such that updating a schedule does not lead to frame loss or additional update overhead. In contrast to [28] and [57], their approach is not limited to TSN and streams are scheduled one by one. Schedules of streams from previous iterations are fixed in later iterations. When some stream cannot be scheduled, backtracking is used by removing some stream of an earlier iteration from the schedule. The authors prove that a set of additional constraints of the ILP imply no conflicts during schedule updates. They evaluate their algorithm with respect to frame loss during updates and update duration on real-world train and automotive networks. The results confirm that no frames are lost and no time overhead is needed for schedule updates.

Another algorithm for schedule updates is proposed by Wang et al. [78]. They present a heuristic scheduling algorithm with backtracking similar to [68]. Additionally, they present an algorithm for incremental schedule updates which omits frame loss during updates. A comparison between both algorithms shows that the incremental update algorithm is faster while it has poor schedulability for higher network utilization.

Gärtner et al. [79] introduce a measure for schedule flexibility denoted as *flexcurve*. The flexcurve is a function that captures the number of possible embeddings of a stream in an existing schedule. Thus, higher values correspond to more possibilities to reschedule a stream. Already scheduled streams may be selected with this measure and shifted to other times to introduce gaps for new streams into a schedule. The paper elaborates on the details of computing and updating

the flexcurves. The authors compare their algorithm to a not specified SMT approach and the algorithm of Santos et al. [59]. In contrast to the SMT approach, solving time of the proposed reconfiguration algorithm is linear for up to 100 streams in the evaluation scenario. The approach of Santos et al. [59] results in schedules with lower flexibility and thus is less suitable for dynamic reconfiguration. A journal extension of this work is presented in [80].

6) GCL SYNTHESIS

Most research works use a post-processing to compute GCLs from transmission offsets. However, this comes with the drawback that GCLs have limited size in bridges and a schedule may not be deployable. We present literature which discusses explicit GCL generation.

Jin et al. [30] present an SMT approach to schedule TSN streams with a fixed number of gate openings. Reducing the number of gate events also reduces the number of guard bands, such that more bandwidth is available for lower-priority traffic. Their modelling assumptions regarding queuing are even more restrictive than frame isolation as only exactly one frame is allowed to be in a queue at any given time. Their approach allows multiple queues for TT traffic per egress port, but assigning streams to queues is not part of the SMT model. This is done before solving the SMT model by a greedy heuristic which aims to balance the workload of all queues of an egress port. As their SMT model cannot be solved in reasonable time, they use an incremental scheme to schedule small groups of streams separately. Subsets of streams are scheduled one after another such that the schedules of previously scheduled subsets are fixed in later iterations. The objective when optimizing a subset is to minimize the maximal number of GCL entries for all egress ports. They also propose a heuristic algorithm which complies with a limited number of GCL entries. Their evaluations show that the heuristic algorithm is an order of magnitude faster than naïve heuristics while reducing the number of GCL entries considerably. Instances with up to 10000 streams were scheduled in reasonable time while the SMT approach has not produced a feasible schedule for an instance with 100 streams within 2 days.

Another incremental SMT scheme which aims to reduce the number of GCL entries is given in [81]. Their approach divides the hyperperiod into slices. Streams and GCLs are scheduled for every slice individually. GCLs are updated and deployed at the beginning of every slice during schedule execution. The authors compare the number of GCL entries needed with schedules computed for an entire hyperperiod. Their results demonstrate that the number of GCL entries can be reduced while keeping end-to-end delays in reasonable bounds. However, it is not a surprise that fewer GCL entries are required when updating the GCLs regularly is allowed, as even a single entry per GCL is sufficient with frequent updates.

A rather simple CP model for scheduling on a single link with only four types of constraints is presented in [82]. However, they propose a post-processing to reduce the number of GCL entries needed in a schedule. For a small test case of only three streams, the authors report a reduction of bandwidth loss due to guard bands by 42.8%.

7) TASK SCHEDULING

Tasks are applications running on end stations. We highlight publications which consider the scheduling of tasks on end stations, additionally to scheduling data streams between these tasks.

In [83], Feng et al. compute schedules for streams and tasks sending or receiving streams simultaneously. The model includes dependencies between streams and tasks, e.g., an application can only be executed when all frames of some stream were received. The authors scheduled instances with 11 streams and more than 100 tasks. As many other works, the authors note the exponential increase of solving times for larger instances.

The authors of [84] present a CP model for scheduling of TT traffic which takes characteristics of control applications into account. Control applications have an execution interval and can only produce output streams for actuators when certain input streams of sensors have arrived. Although the quality of control application execution covers multiple aspects, the only one taken into account is jitter. Queuing is allowed in their model, but is restricted to frame isolation. They compare exact and heuristic search strategies to find solutions to the proposed CP model. For all presented test cases, both search strategies find the optimal solution with zero jitter, but the heuristic approach is orders of magnitude faster.

These preliminary works were extended in [85]. In contrast to [84], a more realistic quality measure for streams of control applications is integrated into the CP model. It constitutes of jitter and end-to-end delays of input and output streams of control applications, and jitter for control application execution. They compare their model with the model from [29] which is extended to include stream precedence for input and output streams of control applications. That means the model is able to enforce that control applications are executed after their respective input streams have arrived. Analogously, streams sent by a control application are scheduled to be transmitted after the execution of the application has finished. The authors report that the presented model outperforms the model from [29] with respect to the proposed quality measure by up to a factor of two on the test cases under consideration. Additionally, they compute a schedule for a realistic test case of an automotive mobile robot and validate their algorithm on a simulation platform and on real hardware. The PhD thesis of Barzegaran [86] features this work.

McLean et al. [87] present a converged approach for task and message scheduling in automotive environments using TSN. The authors propose metaheuristics based on genetic algorithms and simulated annealing to compute the mapping of tasks to processing cores. A combination of list scheduling

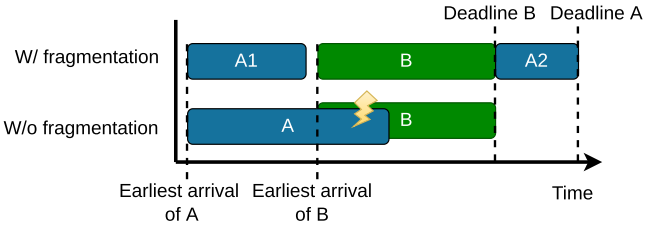


FIGURE 14. Transmissions of two frames A and B over a link to a listener. B cannot be scheduled to be transmitted at another time. In this case, scheduling is only possible when the scheduler splits A into two frames.

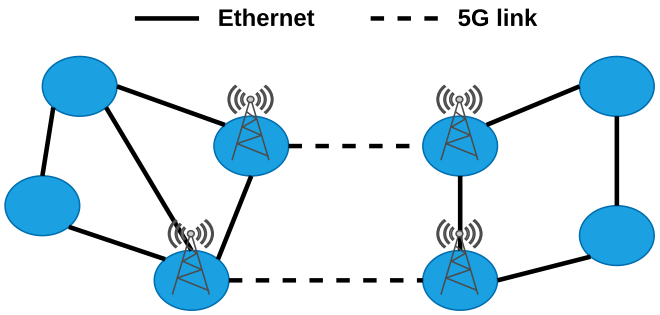


FIGURE 15. Example of a converged network with 5G and Ethernet links as considered in [91].

and earliest deadline first scheduling is used to compute schedules for message transmissions and task executions. Instead of rejecting solutions violating one or more timing constraints, the objective penalizes such solutions. Thus, the algorithm is able to move towards feasible solutions when fining an initial feasible solution is hard. The evaluations show that the simulated annealing algorithm results in lower solving times and better solutions compared to the genetic algorithm heuristic.

Another converged approach for task and message scheduling is proposed by Arestova et al. [88]. The authors introduce the concept of earliest and latest start times for tasks in cause-effect chains of tasks and streams. These times represent the range of valid execution times for tasks and are used for the fast rejection of invalid solutions. An incremental scheduling heuristic which uses these times is presented. Additionally, a repair function is integrated to recover from cases where timing constraints are violated. The evaluations demonstrate that the approach results in significantly reduced solving times and worst case response times compared to approaches from the related work.

8) OTHER TOPICS

This section summarizes research works with unique topics that fit not well into the previous groups.

Jin et al. propose an SMT model which also handles an optimized fragmentation of messages in [89]. Messages can be transmitted with multiple frames. How messages are split into frames is an additional degree of freedom in the presented optimization. Due to performance reasons when solving the model, they also give heuristics for message fragmentation and scheduling. The presented evaluations demonstrate that schedulability increases considerably by up to 50% when message fragmentation is also taken into account. Figure 14 depicts an example of how schedulability can benefit from message fragmentation. Additionally, the presented heuristic algorithms can schedule instances an order of magnitude larger than the SMT approach.

A genetic algorithm approach which takes frame preemption into account is presented in [90]. Their model contains different kinds of MAC interfaces for preemptable and non-preemptable frames. Consequently, the presented synthesis problem not only covers the assignment of streams to queues, but also the assignment of queues to interfaces. Queues are strictly prioritized, i.e., frames contained in a higher-priority queue always preempt frames of a lower-priority queue. The proposed GA aims to maximize the reliability of a schedule. Reliability of a stream is defined as the maximum number of allowed retransmissions without missing the deadline, and the reliability of a schedule is the minimum reliability of all streams. The authors present a comparison of the proposed GA with well-known approaches from automotive traffic scheduling. The baseline approaches are outperformed with respect to schedulability and reliability. The authors explain this result with the fact that their algorithm is specifically constructed to use all the available TSN queues and to utilize them in a way suitable for preemption.

The authors of [91] give a CP model for TSN in joint converged wired and wireless networks. Their model integrates Ethernet and 5G links simultaneously. Figure 15 depicts an example for such a converged network. Frames transmitted over a 5G link must be scheduled to fit into predefined transmission slots. They aim to minimize unusable resources in both types of links, i.e., time occupied by guard bands for Ethernet links and unused bandwidth resources for 5G links. The presented evaluations indicate that minimizing only one kind of unusable resources leads to unsatisfying results for the respective other kind.

Lin et al. [92] evaluate the impact of the so-called *network cycle* to schedulability. The network cycle is a design methodology for frame schedules. All stream periods are assumed to be integer multiples of the network cycle. Frame transmissions are aligned with network cycles. The rational of this is to omit conflicts between streams with different periods when streams are scheduled incrementally. They propose an incremental heuristic which considers the network cycle. The authors report the highest schedulability when the network cycle is set to the greatest common divisor of all stream periods.

The authors of [93] developed a graphical modelling tool for TSN scheduling. They use logic programming to deduce facts about the given problem instance. These facts are in term used for constraint generation of an SMT model. If an instance is infeasible, the conflict refinement capabilities of the SMT solver is leveraged to guide the user in changing the network configuration appropriately. Three test scenarios

are presented where the streams causing infeasibility are identified.

Machine learning techniques were introduced to the domain of TSN scheduling in [94]. Tu et al. present a semi-supervised machine learning model to partition streams in groups before scheduling. They compare their approach with the partitionings in [71] and [95], and state that they are outperformed regarding schedulability. However, it is not clear how this statement is backed by the actual computation of schedules with the resulting stream groups.

We highlighted the contributions of research works for the scheduling problem with fixed routings. We compare and discuss the research works presented in this section together with the research works for the joint routing problem in Section V.

B. SCHEDULING W/JOINT ROUTING

In this section, we give an overview of research work which inspects the joint routing and scheduling problem. In contrast to works in IV-A, algorithms proposed by publications in this section compute a routing and a schedule for a given set of streams simultaneously. Again, we group the literature based on the main topic of the respective papers.

1) JOINT SCHEDULING AND ROUTING W/O PROBLEM EXTENSIONS

This section compiles publications which handle the joint routing and scheduling problem. Research works are only included when they do not focus on an additional topic highlighted in this survey.

An early ILP model which addresses the problem of joint routing and scheduling is presented in [8]. Although it is not exclusively for TSN, the authors state it is applicable for such networks. Their evaluations show that schedulability increases considerably compared to the same test cases with a fixed routing. They compare the solving time of their ILP for joint routing and scheduling with ILPs solely for scheduling. As expected, the solving time is larger for joint routing and scheduling compared to scheduling with a fixed routing. Nevertheless, they still recommend joint routing as solution quality increases considerably.

Falk et al. [96] extend the ILP from [28] to simultaneously compute routing and schedule of TT streams. They analyze the scalability of the joint routing and scheduling problem using ILPs. The authors report that solving time is more influenced by the number of streams than the size of the network topology for their ILP. The evaluations show that network topologies with more paths between any pair of nodes tend to yield harder problem instances, as more routings are possible for any stream.

Nie et al. [97] schedule and route streams incrementally. Streams are grouped by divisibility of their periods, such that streams in the same group can share the same links. The authors focus completely on the case of large period differences and omit the worst case of all streams having the same period. In contrast to similar works, e.g., [71] and [98], they consider only no-wait scheduling. Time is divided into time slots whose lengths equal the greatest common divisor of the periods of all streams. Although many evaluations are performed for different network topologies, network sizes, and traffic types, no results not seen in other works were presented.

Xu et al. [99] propose an incremental SMT scheme similar to [71] and [98]. However, they partition streams with machine learning using some of the ideas from [94]. The authors compare this partitioning approach with the partitioning algorithms from [71], [94], [95], and [100]. The best schedulability was obtained for the proposed partitioning method, second to the methods from [71] and [94]. Schedulability is slightly increased when more streams are scheduled simultaneously, as more conflicting streams are handled in the same iteration. Additionally, the authors compare the incremental scheme with their global scheduling approach from [101]. The incremental scheme outperforms the global approach with respect to schedulability and scalability. The difference in schedulability between both methods increases for higher link utilization.

The authors of [102] present a PBO model for joint routing and scheduling. They compare the solving time of their PB approach with the solving time when routing and scheduling are computed in separate steps by the same model. Their initial evaluations indicate that solving routing and scheduling in separate steps reduces the overall solving time. Surprisingly, their evaluations also show that the 2-step approach performs significantly worse for larger instances compared to the joint approach. They explain this behavior by the capability of SAT solvers to learn from conflicts. Whenever the solver runs into a conflicting variable assignment, it interferes the cause of the conflict and adds a clause to the model which prevents the conflict explicitly. The learned clauses are dropped after the routing step in the 2-step approach. Schedulability increases when routing and scheduling are performed in a single step. They state that instances with more routing options lead to easier solvable scheduling problems as streams can be distributed over the network.

Arestova et al. [103] construct a genetic algorithm for joint routing and scheduling. In contrast to other works with genetic algorithms [100], [104], the authors focus on elaborating on the construction for such an approach in detail. They combine the genetic algorithm with a neighborhood search heuristic to find better solutions efficiently. They allow queuing with flow isolation constraints from [29]. Additionally, a schedule compression algorithm similar to the one in [28] is presented, which is used to reduce the number of guard bands. In a brief evaluation section, they compare their approach with the well-known NEH algorithm [105] from job-shop scheduling. The proposed approach finds feasible schedules faster, while the resulting schedules have comparable flowspans. The authors report that scheduling with joint

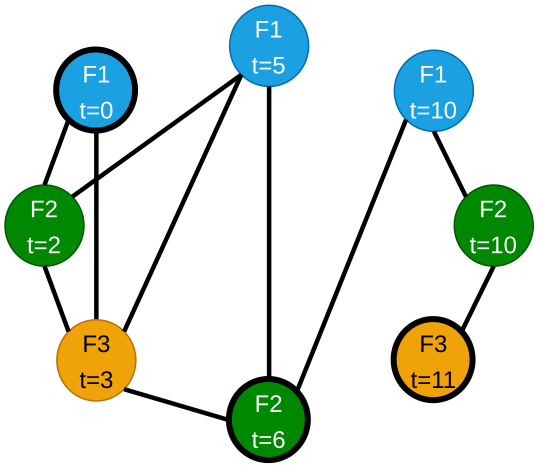


FIGURE 16. Example of a conflict graph used in [108] for a single link topology. Assume the transmission of each frame takes 5 time units. Edges indicate stream schedules which are not compatible. The black circled vertices are an independent set and induce a schedule.

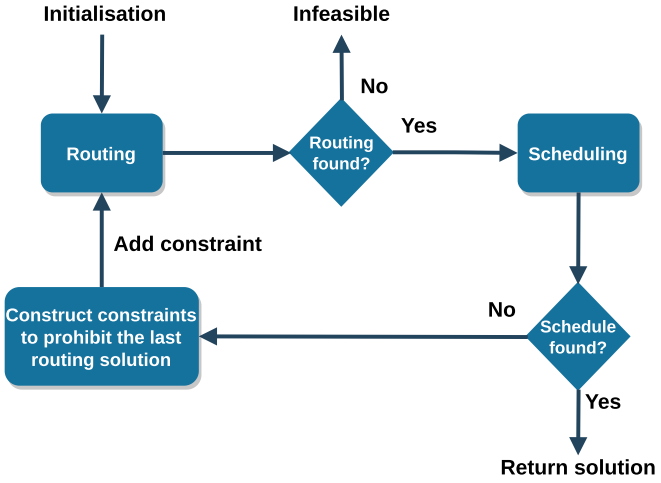


FIGURE 17. Flow diagram of the proposed approach in [109].

routing takes only slightly more time than scheduling with fixed routing with the genetic algorithm.

Kentis et al. [106] investigate the relation of port utilization and GCL schedule duration in a short paper. They employ a simple heuristic for scheduling which is not further explained, and compare the resulting schedule duration with shortest-path routing and the proposed congestion-aware routing. For most test cases, the duration of the schedule is reduced. However, they do not motivate why shorter schedule durations are beneficial.

The authors of [107] present an algorithm based on ant colony optimization. First, they give a heuristic based on simulated annealing and genetic algorithms. The authors state that integrating transmission delays into this approach is hard and propose an ant colony optimization heuristic to overcome this problem. They give the fundamental building blocks of such an algorithm and demonstrate that it can be suitable for TSN scheduling, but also note that further investigation is needed. Unfortunately, they do not elaborate on the routing, and the only related evaluation indicates that increasing the number of edges increases the solving time. The authors compare their TSN-adapted ant colony optimization algorithm with another ant colony optimization that is not specific for TSN scheduling. They conclude that the adapted algorithms results in less jitter and end-to-end delays, and converges after fewer iterations.

Falk et al. propose a joint routing and scheduling algorithm in [108] which is not based on constraints for frame transmissions. Their approach constructs a conflict graph where each vertex represents a schedule for a single stream. Vertices are connected by an edge if and only if the corresponding stream schedules are conflicting. This reduces the scheduling problem to finding an independent set in a conflict graph, i.e., a set of vertices which are pairwise not connected by an edge. Figure 16 depicts an example for a conflict graph and an independent set. The authors use incremental heuristics to construct independent sets in such graphs. Their evaluations demonstrate that the conflict graph approach has advantages regarding runtime and memory consumption compared to ILPs. They remark that their implementation is just a proof of concept, and more efficient algorithms to find independent sets are known. They further note that this approach is cheap, as no expensive ILP solver is needed.

An enhanced CP approach for routing and scheduling is presented by Vlk et al. [109]. The authors present separate models for routing and scheduling, and use them in a problem decomposition algorithm. First, they compute a routing for the given streams. A schedule is computed using this routing. If no schedule was found, constraints are added to the routing model to prohibit the last routing solution. These steps are repeated until a schedule is found. Alternatively, it may be the case that all possible routings for some stream lead to a conflict while scheduling. In that case, an instance is deemed as infeasible. Figure 17 depicts a flow diagram of the proposed approach. Most other research works which performs routing and scheduling in separate steps considers an instance infeasible after only one pass of routing and scheduling. The model also includes queue assignment of streams as additional degrees of freedom. The authors substitute their scheduling model with the algorithms from [28], [29], and [110], and compared schedulability of the resulting algorithms with their approach. Their CP model was able to schedule the most instances, while the SMT model scheduled significantly fewer instances than all other algorithms. Scheduling time was similar for all algorithms except for the SMT model, which needed multiple times longer for most instances.

He et al. [111] present a deep learning based approach for joint routing and scheduling. They use a graph neural network to handle arbitrary sized network topologies. They evaluate their approach on various random network topologies and compare it with [8], [28], [40], and [104]. All other approaches were outperformed in regard to schedulability and scalability for various numbers of streams and network

topology sizes. They also compare different encodings, policies, and sampling strategies featured in their deep learning approach. The results give useful insights for future works involving deep learning in TSN scheduling. Additionally, the authors present measurements of jitter on a real hardware testbed which integrates their deep learning scheduling algorithm. They report no frame losses and ultra low jitter, which indicates that the constructed schedules are valid.

2) SCHEDULING W/RELIABILITY

Reliability is an important topic in the literature of joint routing and scheduling. The possibility of selecting disjoint paths for redundant stream copies further increases reliability in these research works.

Pozo et al. [112] present an ILP for joint routing which considers reliability constraints. After a single link failure, all streams over this link must be rescheduled and rerouted. The authors propose a fast heuristic for this case. They also evaluate which properties of a schedule are beneficial for the repairability in case of a failure. The results indicate that schedules which maximize the gaps between frame transmissions are much easier to repair than schedules which minimize the flowspan, even for three simultaneous link failures.

Atallah et al. [113] give a heuristic for fault-tolerant joint scheduling, routing, and topology generation. Their algorithm starts with a full mesh topology. Streams are routed and scheduled one after another. A k-shortest-paths algorithm is used to enumerate possible paths for a stream. If a path with available time slots is found, schedule and routing are fixed for later iterations. This is repeated multiple times with disjoint paths for redundant copies of a stream to ensure reliability. Links and bridges are only included in the final topology when they are used by some stream. The algorithm also selects bridges such that more expensive bridges are only used when necessary. Figure 18 depicts this approach for topology generation. The authors compare their algorithm to another approach which realizes redundant paths through multiple copies of the network topology. The proposed algorithm scheduled all problem instances, while it reduces financial costs for network hardware considerably.

The authors of [71] propose an incremental ILP scheme to comply with requirements for the robustness against single link failures. First, the authors present a GRASP heuristic for routing which considers reliability constraints. Streams are replicated and routed over disjoint paths to comply with requirements regarding robustness to single link failures. The resulting routing is used as input to the incremental ILP approach. Streams are partitioned into groups by introducing a conflict metric and computing a weighted cut in the conflict graph. Groups of streams are scheduled one after another. The computed schedules are fixed when the next group is scheduled. They model egress ports with only one queue for TT traffic and frame isolation. A comparison of the presented ILP with the ILPs from [8] and [102] demonstrates that it outperforms both approaches regarding solving times.

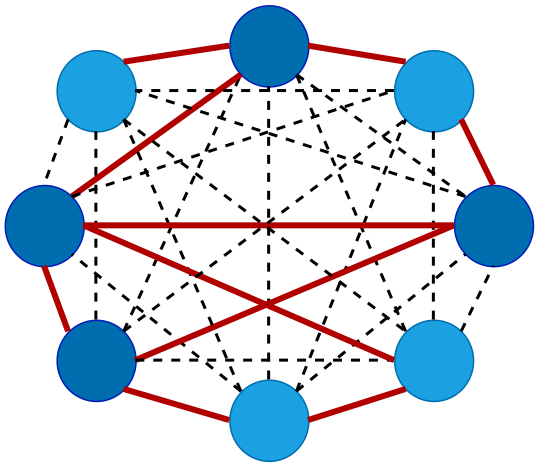


FIGURE 18. Topology generation approach of [113]. A full mesh topology is used during routing and scheduling. Links used in the final routing are indicated in red. Only these links are included in the final topology. The approach covers the selection of bridges from a library depending on the generated topology. More expensive bridges with a higher number of ports are indicated in dark blue.

Instead of robustness against link failures, Zhou et al. [114] consider reliability against frame loss, i.e., frames that are lost spontaneously during transmission without a permanent link failure. They integrate constraints regarding the loss probability along the routed path of a stream in their SMT model. However, these probabilities are only approximated, as the used theory solver cannot handle exponentials. An incremental scheme similar to [71] is used. Subsets of streams are scheduled and routed one after another until all streams are scheduled. Stream schedules are fixed in subsequent iterations. They use redundant copies of streams to further reduce the probability of frame loss, as there may be no single path with the required reliability. In contrast to other works, e.g., [71], [72], and [73], they do not enforce paths to be link disjoint. Their evaluations show that schedulability with a given level of reliability against frame loss increases with a higher number of redundant copies per stream.

Syed et al. [115] present an ILP and a heuristic for scheduling and path selection in in-vehicle networks. They leverage Frame Replication and Elimination for Reliability (FRER) [17] to ensure robustness against frame loss of safety-critical streams. The ILP model is similar to the ILP used in prior works by the same authors [116]. They report solving times of about a day with the ILP while the heuristic solved the same instances in a few minutes.

Following their works in [115], Syed et al. [117] developed an alternative to FRER. The authors propose a network coding scheme to mitigate temporary and permanent link failures. Two disjoint paths are used to transmit two frames. A third path disjoint from the other two is used to transmit the XORed data of these two frames. The loss of one frames can be tolerated as the lost frame can be reconstructed from the other two. Therefore, the redundant transmission of n frames results in $\frac{3}{2n}$ frame transmission with this scheme. This is a

significant reduction compared to the $2n$ frames needed with FRER.

Another incremental scheme for scheduling and routing in safety-critical automotive applications is presented by Zhou et al. [118]. They consider possibly undetected systematic faults of bridges, i.e., implementation bugs or divergence from specifications, instead of randomly arising errors like frame loss. Messages and bridges have an Automotive Safety Integrity Level (ASIL) assigned [119] which defines reliability constraints of the respective component. A higher ASIL corresponds to higher reliability. Additionally, to computing routing and scheduling, their algorithm also selects which bridges to use from a library. Messages can be decomposed into redundant message copies with lower ASIL to be sent over bridges with lower ASIL. A comparison of the presented algorithm with and without message decomposition shows that the total financial costs for bridges can be reduced by up to 23.55% in the evaluation scenarios. The authors note that higher ASILs increase the required number of message copies, which leads to more congestion in the network and thus higher end-to-end delays. Evaluations on a real-life automotive test case are presented. Synthesis time increases considerably with ASIL decomposition. However, the selected bridges cost only a third compared to using only bridges with the highest ASIL. The PhD thesis of Zhou [120] compiles the content of [114], [118], and [121].

The authors of [72] present separate CP models to compute schedules and routings as work-in-progress. These models take security and reliability considerations into account. The routing computed with the first model is used as fixed input for the scheduling model, similar to [71]. Redundant streams with disjoint paths are added if needed to comply with security and reliability constraints.

The ideas from [72] are extended in [55]. Besides the CP model, a simulated annealing algorithm combined with a list scheduler is given. Additionally, a post-processing for latency reduction of scheduled streams is applied after scheduling. Their evaluations without security and reliability constraints indicate that the SA approach is able to find a feasible schedule in reasonable time, even for huge problem instances. However, they also introduce a measure for schedule and routing costs. This measure contains stream latencies, penalties for streams which were not scheduled, and penalties for overlapping paths. The results demonstrates that schedules computed by SA have up to three times higher costs compared to schedules computed by the CP approach. Introducing the security and reliability constraints to the same problem instances increased the costs of schedules computed by SA by up to a factor of ~ 3.5. Nevertheless, the authors state that the SA approach can be useful in comparing costs and reliability capabilities of topologies, or to reconfigure the network in case of link failures.

Li et al. [122] propose a heuristic for joint routing and scheduling with reliability constraints. A greedy algorithm is used to select paths for redundant copies of streams such that link utilization is balanced. Frames are scheduled as soon as possible. Both algorithms are combined in an iterative local search scheme. Already scheduled and routed streams are randomly removed from time to time, and the remaining streams are rescheduled and rerouted in random order. While the authors report a reduction in frame losses by their approach, end-to-end delays are significantly larger compared to schedules with routings computed by Dijkstra's algorithm.

3) SCHEDULING W/OTHER TRAFFIC

This section presents research works concerned with scheduling in the presence of other traffic classes.

Gavriluţ et al. [123] construct a GRASP heuristic to schedule TT streams while taking AVB traffic into account. In contrast to [57], their approach handles TT and AVB streams simultaneously. The routing of AVB streams is given and cannot be changed. The problem of finding a routing for AVB streams in the presence of TT streams was addressed and evaluated by Laursen et al. [124]. The authors of [124] propose a hill climbing based heuristic algorithm. In the first evaluation of [123], the authors use a shortest paths algorithm to compute the routing for TT streams. The results show that using the GRASP heuristic with an objective that considers tardiness of AVB streams leads to AVB streams meeting their deadlines. In contrast to that, the GRASP heuristic with other objectives not considering AVB streams results in schedules with late AVB streams. Even better results are obtained when a routing with load balancing is used before scheduling, which also decreases overall runtime. They report that AVB traffic does not benefit from minimizing the number of queues for TT streams. However, this is rather obvious, as their system model assumes that AVB streams already have an assignment to AVB queues. Solving time of the GRASP heuristic increases considerably when AVB streams are taken into account. A short preview of these results was previously published in [125] and the implementation details of the heuristic are elaborated in [126].

The routing algorithm in [124] only considers AVB streams in an offline scenario. Another work which focuses on the routing of AVB streams in the presence of scheduled traffic is presented in [127]. The authors propose an online routing algorithm for AVB and TSN streams. The algorithm is based on ant colony optimization and compared to the approach of [124]. The evaluations indicate that the ant colony optimization algorithm outperforms the approach of [124] with respect to solving times.

Gavriluţ et al. [128] propose an algorithm which assigns messages to traffic classes, i.e., whether a message should be transmitted with TT or AVB streams and which AVB class should be used. The assignment algorithm is based on the tabu search metaheuristic. Additionally, the parameters for the CBS are estimated such that AVB streams comply with their real-time requirements. Afterwards, streams are scheduled

with an adaption of the algorithm from [123] and [126]. The evaluations show that the assignment algorithm significantly increases the number of schedulable streams. Compared to the SMT approach of [29], the assignment to traffic classes increases the schedulability of legacy streams that cannot be reassigned or rescheduled.

Berisa et al. [129] propose a heuristic for the joint routing and scheduling of TT streams in the presence of AVB streams. They make use of frame preemption to increase the schedulability of AVB streams. To that end, they present a worst-case response-time analysis of AVB streams with preemption. The heuristic is based on prior works by Gavriluţ et al. [123]. Their evaluations demonstrate that the schedulability of AVB streams can be increased by allowing frame preemption while the runtime of the heuristic also increases significantly in this case.

Alnajin et al. [32] give a QoS-aware routing algorithm for TSN streams with respect to various metrics. They present four scheduling heuristics combined with these routings. They compare these algorithms regarding the number of guard bands needed in the resulting schedules. Their evaluations show that their heuristics can reduce the number of guard bands significantly. They note that reducing the number of guard bands is beneficial for BE traffic.

Li et al. [130] present a heuristic for joint routing and scheduling to eliminate non-deterministic queuing delay in networks with mixed time-critical traffic. Their scheduling algorithm divides the bandwidth resources into time slots and assigns streams to these slots such that transmission conflicts cannot arise. Similar to [31], the maximum length of the resulting GCLs is bounded instead of being computed by a post-processing from transmission offsets. The solving time with the heuristic is compared to [71], [96], and [103]. Solving time is the only inspected metric as the presented algorithm, and the three compared approaches have no objective functions. In the presented scenarios, all three methods were outperformed by multiple orders of magnitude. The authors report schedules for 4000 streams with just 12 GCL entries per egress port on average.

Yang et al. [131] use deep reinforcement learning for the joint routing and scheduling problem. Additionally, they take AVB and BE streams into account. The authors elaborate on the details of their machine learning model and present evaluations about the learning phase. They introduce three baseline approaches also based on machine learning for comparison. The proposed model results in slightly lower average latencies for all traffic classes in the evaluation scenario.

4) MULTICAST

This section highlights research works specifically concerned with multicast streams in a joint routing and scheduling approach.

The joint routing and scheduling model from [8] is extended for multicast support in [132]. The authors state that while this extension is trivial for pure scheduling models, joint routing and scheduling with multicast is more complicated as additional constraints for the routing are needed. Various pre-processing steps are presented to reduce the solution space and thus solving time. The authors report that the time to find a feasible solution was reduced by up to 82.4% while the overall solving time was reduced by up to 47.6%.

Another approach for joint routing and scheduling with multicast streams is presented by Li et al. [133]. Similar to [132], the authors use pre-processing to simplify solving of the model. The streams are divided into groups by spectral clustering based on their properties. Similar to [71] and [72], these groups are routed and scheduled one after another such that previously computed schedules and routes are fixed. The authors report that random clustering result in slightly longer flowspans. As in similar incremental approaches, reduced overall solving times and increased schedulability are reported.

Yu et al. [134] propose an incremental approach with ILPs. In contrast to [132], they route and schedule multicast streams one by one. Multiple queues per egress port and queue assignment are also integrated in their model. Additionally, they propose a pre-processing scheme which aims to simplify the topology. The pre-processing merges cliques in the topology to a single link. If routing and schedule can be computed, both are modified for the original graph. Otherwise the conflicting links are expanded and routing and scheduling are repeated. Compared to [7] with a Steiner tree as fixed routing, the proposed approach can schedule significantly more instances.

A biology-inspired algorithm is given by Pahlevan et al. [104]. They construct a genetic algorithm for joint routing and scheduling which also comprises features like multicast streams and dependencies between streams. The authors state multicast capabilities as one of their main contributions, but consider multicast streams simply as multiple unicast streams. In contrast to [29], [31], and [125], only a single queue per egress port is dedicated to TT traffic. While their evaluations indicate that solving time increases compared to scheduling with a fixed routing, they show that schedulability increases by joint routing and scheduling.

In later works, Pahlevan et al. [100] present a heuristic list scheduling algorithm for the same purpose. They model queuing and multicast streams in the same way as in [104]. Their evaluations again demonstrate that joint routing and scheduling increases schedulability.

5) RECONFIGURATION OF SCHEDULES

Reconfiguration of streams can benefit from modifying not only a schedule, but also the respective routing. Newly added streams can be routed over paths with low utilization. This section compiles the literature about reconfiguration in joint routing and scheduling apporaches.

Research work from Syed et al. [116] deals with joint routing and scheduling in in-vehicle networks. They propose an ILP model for streams that are known in advance which takes load balancing into account. They compute schedules

and routings for these streams such that as many streams as possible can later be added dynamically. The evaluations compare algorithmic details that are hard to asses without detailed knowledge of the presented approach.

Following their work in [116], Syed et al. present multiple heuristics for the dynamic scheduling of new streams in an existing schedule [135]. Their heuristics are based on modelling the scheduling problem as a vector bin packing problem. They evaluate the time needed for adding new streams in an automotive use case, as reconfiguration in such scenarios has strict timing requirements.

The authors of [33] construct multiple heuristics for dynamic scheduling and routing with reliability constraints. All heuristics are based on the same idea as in [135]. Every stream is replicated twice when added to an existing schedule to ensure reliability. The best heuristic is able to schedule 500 streams in about 410 ms. The authors state this is a reasonable response time in automotive use cases.

Yu et al. [136] developed a heuristic for online rescheduling in scenarios with virtual machines as communication endpoints. Virtual machines in a cloud computing environment may be migrated from one physical device to another such that schedules and routings must be updated. Additionally, all streams are multicast streams, which complicates rescheduling after a VM migration. Therefore, the multicast tree for a stream is computed such that the maximal distance from any possible device where a VM could run to any destination is minimized. The authors state that this will reduce conflicts when a VM is migrated, as the new paths are short. Given a schedule and a stream that is migrated, a greedy heuristic computes the new schedule based on the precomputed multicast tree. The authors compare their proposed routing heuristic with an optimal routing obtained by an ILP. Solving times are significantly reduced, while the routing objective grows only slightly compared to the optimal routing. Schedulability in case of a migration is considerably increased in comparison to the same scheduling heuristic used with a routing computed by the KMB algorithm [137].

Li et al. [138] consider the reconfiguration of routing, scheduling, and mapping of applications to end stations in case of permanent end station failure. They extend the ILP of [139] to schedule applications to end stations for global reconfiguration. As the resulting ILP instances are hard to solve, they propose a heuristic routing and mapping algorithm as alternative. The results of this heuristic are fixed in the ILP model, such that only a schedule is computed. The heuristic approach is able to reconfigure almost all instances, while the ILP times out for most of them in their evaluations. While both algorithms have exponential runtime in the number of streams, the heuristic is two orders of magnitude faster for the considered instances.

An incremental approach which schedules streams one by one is presented in [98]. In contrast to [134], the computed schedules are constrained to no-wait scheduling, i.e., no queuing delays are allowed. The authors compare the proposed approach with [8] and [71] with respect to schedulability and show that schedulability is slightly increased. The proposed pre-processing for the routing approach gives only minor improvements regarding schedulability. The authors report that 97.5% of the streams in an instance with 2000 streams were scheduled in less than 10 seconds per stream. They state this is fast enough for online scenarios.

6) OTHER TOPICS

This sections summarizes research works with unique topics that do not fit well into the previous groups.

The authors of [121] propose a heuristic model to schedule streams in the presence of frame preemption similar to [90]. Additionally, they also include route computation in their algorithm. They present an SMT model for this purpose and use it in an incremental approach, similar to [95], [102], and [140]. The presented results show that scheduling time not only increases with the network size and the number of streams, but also with the maximum number of allowed preemptions and retransmissions. However, allowing more preemptions increases schedulability only to some instance-specific threshold.

Gavriluţ et al. [73] give multiple algorithms to simultaneously compute scheduling and routing of TT streams. In contrast to [71], [72], and [96], these algorithms additionally generate the network topology with minimal financial costs imposed by network hardware. They present a problem-specific heuristic, a GRASP heuristic, and a CP approach, and compare them to each other regarding solving time and solution quality. Their optimization objective captures worst-case end-to-end delays as well as topology costs, i.e., costs for links and bridges which are selected from a library. Redundant copies of streams are included for reliability considerations. Their evaluations focus on a comparison of the three presented algorithms. As expected, the CP approach does not scale well. The GRASP heuristic finds better solutions in minutes compared to the CP approach in two days.

An SMT model which includes scheduling, routing, and queue assignment of streams simultaneously is presented by [101]. The authors state that saving bandwidth by not using the same GCL cycle for all egress ports is also novel to their approach. However, this is not true as other works even schedule GCL closing events, e.g. [30] and [41]. They propose to minimize the number of bridges used by scheduled traffic in order to maximize utilization. In comparison to the list scheduler of [100], schedulability is increased while the solving time approaches the timeout after 40 h for fairly small instances.

Zhang et al. [141] construct a heuristic which allows different routes for frames of the same stream to enable load balancing. The required mechanism is implemented by an SDN architecture. The scheduling procedure is a mix of evolutionary algorithm and greedy algorithm. Multiple variations of the heuristic are compared in the evaluations. In the

presented scenarios, scheduling time increased linearly with the number of streams.

Another incremental scheme for scheduling and routing is presented by Mahfouzi et al. [95], [140]. The authors investigate the stability of control applications, i.e., latency and jitter of messages sent by these applications. Instead of grouping streams by some conflict measure, they divide the network period in slices and group the streams by the time slice in which their transmission can start. They allow routing only over some fixed number of precomputed shortest paths per source-destination pair, similar to [123] and the AVB routing in [57]. Their model allows unrestricted queuing without further discussion of this topic. The evaluations indicate that the number of allowed paths has a huge impact to solving time. More possible paths result in a significant increase in solving times. However, they note that three paths per pair of nodes may be sufficient, as schedulability is over 90% in this case. They conclude that the search space can be reduced without decreasing schedulability considerably.

Yang et al. [142] present a network architecture for industrial use cases based on TSN hardware and software-defined networking (SDN). They focus on so-called chain flows. Chain flows consist of multiple stream which are joined at one or more nodes. For instance, an industrial controller may join streams from multiple sensors and forward a single stream to an actuator. The authors propose a tabu search heuristic and an ILP for the scheduling of chain flows. They report benefits in scalability and schedulability in comparison to handling every stream of a chain flow individually.

Chain flows are further investigated by Gong et al. [143]. They propose a heuristic time-tabling algorithm combined with a tabu search for schedule reconfiguration when the network topology is changed. In contrast to the magazine article [142], they elaborate on the details of these algorithms. However, the reported results are consistent with the results in [142].

Hellmanns et al. [144] focus their work not on the ability to compute schedules, but analyze how input pre-processings and solver configuration influence the scalability of solving a joint routing and scheduling ILP model. They categorize optimizations by whether they are input pre-processing, e.g., topology reduction, model generation related, e.g., tighter variable bounds, or solver configurations, e.g. the use of value hints for variables. They give an ILP without any optimizations as baseline for their evaluations. Different combinations of the proposed optimizations are tested on the same set of problem instances and compared with respect to scalability and schedulability. Their evaluations indicate that solving time can greatly benefit from input pre-processings, but the effects of model generation optimizations and solver configurations are negligible. Some of the optimizations even increase solving time. However, queuing is not supported by their base model. Thus, the observations only hold for the no-wait case without queuing delay. It is not clear whether these results can be transferred to other problem extensions from the literature.

Bhattacharjee et al. [145] propose two algorithms for the placement of talker applications in a network. Additionally, both approaches also solve the joint routing and scheduling problem. The first algorithm is an ILP for the placement of talker applications that is combined with the GRASP heuristic of [123]. The second algorithm is a simulated annealing (SA) heuristic which computes the placement of talker applications, the schedule, and the routing for a given problem instance. The evaluations show that both algorithms behave approximately similar with respect to load balancing and solving times. However, the authors report considerably reduces stream latencies with the SA heuristic.

V. COMPARISON AND DISCUSSION

We compare the presented research work from Section IV. First, we compile modelling assumptions and problem extensions. Second, we present common scheduling objectives. Then, we investigate problem instances used for evaluations. Finally, we summarize results regarding the scalability of the presented approaches.

A. MODELLING ASSUMPTIONS AND PROBLEM EXTENSIONS

Table 2 compiles important modelling assumptions and problem extensions in the surveyed research works with fixed routings. Table 3 shows the same information for research works about the joint routing problem. In the following section, we compile the contributions to each of these topics.

1) OTHER TRAFFIC

Only five works examine TT and AVB streams simultaneously. Pop et al. [57] present a GRASP heuristic for the handling of AVB streams. The heuristic gets a schedule of TT streams as input and cannot change it. The authors of [125] present a short preview of AVB-aware scheduling, which was later extended in [123]. An adaption of this approach was used in [128]. The authors present an algorithm to assign messages to traffic classes in networks supporting AVB and TSN simultaneously. Feng et al. [83] consider the bandwidth available to AVB and BE traffic in their approach as they consider repeated frame loss which may result in starvation. Berisa et al. [129] use frame preemption and a worst-case end-to-end delay analysis to increase the schedulability of AVB streams. Huang et al. [68] give parameters for AVB streams in an in-vehicle network and include them in their evaluation scenario. Wang et al. [67] consider the joint handling of AVB and TT streams. AVB streams are shaped by CQF. Their objective aims to reduce the influence of non-periodic BE traffic to AVB streams. Some works focus solely on the routing of AVB streams in the presence of scheduled traffic in TSN [124], [127]. Li et al. [58] present a heuristic to configure the CBS in the presence of scheduled traffic.

TABLE 2. Overview of considered problem extensions and restrictions in the literature of scheduling approaches with a fixed routing.

Research work	AVB	BE	Queuing	Fixed GCL length	Reconfiguration	Reliability	Multicast	Task scheduling
Ansah *et al.* [45]	-	-	✓	-	-	-	-	-
Arestova *et al.* [88]	-	-	✓	-	-	-	-	✓
Barzegaran *et al.* [85]	-	-	(✓)	-	-	-	✓	✓
Barzegaran *et al.* [64]	-	(✓)	✓	✓	-	-	-	-
Barzegaran *et al.* [84]	-	-	(✓)	-	-	-	-	✓
Bujosa *et al.* [54]	-	-	✓	-	-	-	✓	-
Chaine *et al.* [53]	-	-	(✓)	-	-	-	-	-
Craciunas *et al.* [29]	-	-	(✓)	-	-	-	-	-
Craciunas *et al.* [69]	-	-	(✓)	-	-	✓	-	-
Dai *et al.* [82]	-	(✓)	-	-	-	-	-	-
Dobrin *et al.* [75]	-	-	✓	-	-	✓	-	-
Duerr *et al.* [28]	-	✓	-	-	-	-	-	-
Farzaneh *et al.* [93]	-	-	-	-	-	-	-	-
Feng *et al.* [70]	✓	✓	(✓)	-	-	✓	-	-
Feng *et al.* [74]	-	✓	✓	-	-	✓	-	-
Feng *et al.* [83]	-	-	(✓)	-	-	-	✓	✓
Gärtner *et al.* [79][80]	-	-	(✓)	-	✓	-	-	-
Gavriluţ *et al.* [125]	✓	-	(✓)	-	-	-	-	-
Gavriluţ *et al.* [128]	✓	✓	(✓)	-	-	-	✓	-
Ginthör *et al.* [91]	-	✓	✓	✓	-	-	-	-
Hellmanns *et al.* [40]	-	(✓)	(✓)	-	-	-	-	-
Houtan *et al.* [62]	-	✓	(✓)	-	-	-	-	-
Huang *et al.* [68]	✓	(✓)	(✓)	-	-	✓	-	-
Jin *et al.* [89]	-	-	-	-	-	-	-	-
Jin *et al.* [30]	-	(✓)	(✓)	✓	-	-	-	-
Kim *et al.* [42]	-	(✓)	✓	-	-	-	-	-
Kim *et al.* [43][44]	-	(✓)	✓	-	-	-	-	-
Lin *et al.* [92]	-	-	-	-	✓	-	-	-
McLean *et al.* [87]	-	-	(✓)	-	-	-	-	✓
Min *et al.* [47]	-	-	-	-	-	-	-	-
Oliver *et al.* [31]	-	(✓)	(✓)	✓	-	-	✓	-
Pang *et al.* [77]	-	-	✓	-	✓	-	✓	-
Park *et al.* [90]	-	-	✓	-	-	✓	-	-
Pei *et al.* [65]	-	✓	✓	-	-	-	-	-
Pop *et al.* [57]	✓	-	(✓)	-	-	-	Only AVB	-
Raagaard *et al.* [76]	-	-	(✓)	-	✓	-	-	-
Reusch *et al.* [52]	-	(✓)	✓	✓	-	-	-	-
Santos *et al.* [59]	-	✓	✓	✓	-	-	✓	-
Steiner [7]	-	-	✓	-	-	-	✓	-
Steiner *et al.* [41]	-	(✓)	(✓)	✓	-	-	✓	-
Vlk *et al.* [48]	-	-	(✓)	-	-	-	-	-
Vlk *et al.* [51]	-	-	✓	-	-	-	-	-
Wang *et al.* [50]	-	(✓)	-	-	-	-	-	-
Wang *et al.* [46]	-	-	✓	✓	-	-	-	-
Wang *et al.* [78]	-	-	✓	-	✓	-	-	-
Wang *et al.* [67]	✓	✓	-	-	-	-	-	-
Yao *et al.* [66]	-	✓	✓	✓	-	-	-	-
Zhang *et al.* [22]	-	-	-	-	-	-	✓	-
Zhou *et al.* [63]	-	-	(✓)	-	-	-	-	-

Other research works handle BE traffic by minimizing the flowspan which yields a large time slot at the end of a schedule exclusively for other traffic, e.g., [28], [40], and [62]. Pei et al. [65] evaluate their approach in the presence of BE traffic and rate constrained traffic. Yao et al. [66] consider the joint scheduling of periodic stream without real-time requirements in their approach. The tables indicate works that do not mention BE traffic, but use objective functions that are beneficial to other traffic or that limit the number of guard bands with (✓). For instance, Oliver et al. [31] limit the number of guard bands indirectly by introducing a fixed number of transmission time windows per egress port.

2) QUEUING

Queuing is a controversial topic in the TSN scheduling literature as non-determinism, e.g., frame loss, may cause serious problems. Some research works do not allow queuing at all, e.g., [28], [71], and [96]. The majority of the algorithms in the literature uses frame isolation introduces by [29]. These works are indicated by (✓) in the Tables 2 and 3. Vlk et al. [51] discuss the effects of isolation constraints. They report results indicating that isolation constraints reduce schedulability significantly. Without isolation, the number of scheduled streams can be increased for all evaluated topologies and problem instance sizes. Additionally, isolation

TABLE 3. Overview of considered problem extensions and restrictions in the literature of joint routing approaches.

Research work	AVB	BE	Queuing	Fixed GCL length	Reconfiguration	Reliability	Multicast	Task scheduling
Alnajim *et al.* [32]	-	✓	✓	-	-	-	-	-
Arestova *et al.* [103]	-	(✓)	(✓)	-	-	-	-	-
Atallah *et al.* [113]	-	-	-	-	-	✓	-	-
Atallah *et al.* [71]	-	-	(✓)	-	-	✓	-	-
Berisa *et al.* [129]	✓	-	✓	-	-	-	-	-
Bhattacharjee *et al.* [145]	-	-	(✓)	-	-	-	-	-
Falk *et al.* [96]	-	-	-	-	-	-	-	-
Falk *et al.* [108]	-	-	-	-	-	-	-	-
Gavriluţ *et al.* [73]	-	-	-	-	-	✓	✓	-
Gavriluţ *et al.* [123]	✓	-	(✓)	-	-	-	-	-
Gong *et al.* [143]	-	-	✓	-	-	-	-	-
He *et al.* [111]	-	-	✓	-	-	-	-	-
Huang *et al.* [98]	-	-	-	-	✓	-	-	-
Kentis *et al.* [106]	-	-	✓	-	-	-	-	-
Li *et al.* [122]	-	-	✓	-	-	✓	✓	-
Li *et al.* [130]	-	✓	✓	✓	-	-	-	-
Li *et al.* [81]	-		-	-	-	-	-	-
Li *et al.* [138]	-	-	-	-	✓	-	-	-
Li *et al.* [133]	-	-	(✓)	-	-	-	✓	-
Mahfouzi *et al.* [95][140]	-	-	✓	-	-	-	-	✓
Nie *et al.* [97]	-	-	-	-	-	-	-	-
Pahlevan *et al.* [104]	-	(✓)	-	-	-	-	-	✓
Pahlevan *et al.* [100]	-	(✓)	-	-	-	-	-	✓
Pozo *et al.* [112]	-	-	-	-	-	✓	✓	-
Reusch *et al.* [55]	-	-	(✓)	-	-	✓	✓	✓
Reusch *et al.* [72]	-	-	-	-	-	✓	-	✓
Schweissguth *et al.* [8]	-	-	-	-	-	-	-	-
Schweissguth *et al.* [132]	-	-	-	-	-	-	✓	-
Smirnov *et al.* [102]	-	✓	-	-	-	-	✓	-
Syed *et al.* [116]	-	-	-	-	-	-	-	-
Syed *et al.* [33]	-	-	-	-	✓	✓	-	-
Syed *et al.* [135]	-	-	-	-	✓	-	-	-
Syed *et al.* [115]	-	-	-	-	-	✓	-	-
Syed *et al.* [117]	-	-	-	-	-	✓	-	-
Vlk *et al.* [109]	-	-	(✓)	-	-	-	-	-
Wang *et al.* [107]	-	-	-	-	-	-	-	-
Xu *et al.* [101]	-	-	(✓)	✓	-	-	-	-
Xu *et al.* [99]	-	-	(✓)	-	-	-	-	-
Yang *et al.* [142]	-	-	✓	-	-	-	-	-
Yang *et al.* [131]	✓	✓	✓	-	-	-	-	-
Yu *et al.* [134]	-	-	(✓)	✓	-	-	✓	-
Yu *et al.* [136]	-	-	(✓)	-	✓	-	✓	-
Zhou *et al.* [118]	-	-	-	-	-	✓	-	-
Zhou *et al.* [114]	-	-	-	-	-	✓	-	-
Zhou *et al.* [121]	-	-	✓	-	-	-	-	-
Zhou *et al.* [141]	-	-	(✓)	-	-	-	-	-

results in larger runtimes compared to scheduling without isolation. In contrast, the differences in end-to-end delays are negligible. However, they propose a different solution to deal with non-determinism, effectively modifying the TAS. Thus, their results are not applicable to current TSN implementations. There are some research works which allow unrestricted queuing, e.g., [59] and [75]. Most of these works do not elaborate on the consequences of unrestricted queuing. In contrast to that, Reusch et al. [52], Barzegaran et al. [64], and Berisa et al. [129] introduced countermeasures for the mentioned consequences. The authors include a worst-case end-to-end delay analysis in their algorithms such that even in the case of non-determinism, deadlines are met.

3) FIXED GCL LENGTH

Most algorithms presented in the literature handle the generation of GCLs indirectly. They schedule transmission offsets of frames at end stations and intermediate bridges. The GCLs are generated by a post-processing after scheduling. This step is only mentioned, and the respective authors do not elaborate on it. Examples for such works are [29], [48], and [55]. However, computing GCLs by a post-processing

comes with two drawbacks. First, GCLs have limited size in bridges. Thus, GCLs obtained by a post-processing may not be deployable. Second, the scheduling algorithm cannot include considerations for guard bands. There are some exceptions to this in the literature. Jin et al. [30] present a heuristic to compute schedules with a limited number of GCL entries per egress port. Santos et al. [59] give a detailed SMT model for TSN scheduling which includes the explicit representation of GCLs. Yao et al. [66] limit the number of GCL entries and the maximum queue size in their heuristic, i.e., schedules which do not meet these constraints are not considered valid solutions. Some works limit the number of GCL entries indirectly by introducing transmission windows for egress ports. Streams are mapped to these transmission windows and their number is fixed before scheduling. Examples of such works are [31], [41], and [52]. All schedules for no-wait scheduling can be deployed with a fixed number of GCL entries. As no queuing delay is allowed, frames cannot be scheduled to wait at closed gates. Such a schedule can be deployed by opening all gates for TT traffic at the start of a hyperperiod and never closing them.

4) RECONFIGURATION

In some scenarios it may be infeasible to compute new schedules every time a stream should be integrated into or removed from an existing schedule. For instance, automotive scenarios may include ad-hoc connections between cars and infrastructure. Computing new schedules every time a new stream is added may take too much time, even with heuristic algorithms. Syed et al. [33], [135] consider reconfiguration in such automotive scenarios. Additionally, they also present preliminary work about computing schedules suitable for later reconfiguration in [116]. Raagaard et al. [76] present a heuristic to add streams to an existing schedule. When the heuristic fails, they assign the new stream to other egress queues which were unused before. Pang et al. [77] present work about deploying an updated schedule to a network already executing another schedule. Their approach allows updating the schedule without frame loss or new streams interfering with the old schedule. Another use case for reconfiguration is the reallocation of tasks sending and receiving TT streams. Yu et al. [136] use virtual machines as end stations in their model. These virtual machines may be migrated from one physical device to another, which requires updating schedules and routings. A similar example for reconfiguration is presented in [138]. The authors propose an approach for updating a schedule in case of a permanent end station failure. Schedules and routings must be updated in this case as in [136]. Gärtner et al. [79] introduce a measure for schedule flexibility which also considers deadlines. They use this measure to update schedules in a beneficial way for future updates. Lin et al. [92] show in their evaluations that aligning frame transmissions to the greatest common divisor results in schedules that are suitable for adding streams later.

TABLE 4. Fault models in research works dedicated to reliability.

Fault Model	Research work
Permanent link failure	Atallah *et al.* [113], Atallah *et al.* [71], Gavriluţ *et al.* [73], Pozo *et al.* [112], Reusch *et al.* [55], Reusch *et al.* [72], Syed *et al.* [33], Syed *et al.* [117]
Frame loss	Atallah *et al.* [113], Atallah *et al.* [71], Dobrin *et al.* [75], Feng *et al.* [70] [74], Gavriluţ *et al.* [73], Huang *et al.* [68], Li *et al.* [122], Park *et al.* [90], Reusch *et al.* [55], Reusch *et al.* [72], Syed *et al.* [33], Syed *et al.* [115], Syed *et al.* [117], Zhou *et al.* [114]
Clock drift	Craciunas *et al.* [69]
Hardware bugs	Zhou *et al.* [118]

5) RELIABILITY

Table 4 compiles fault models used in the literature. The listed research works construct schedules robust in the respective fault model. Computing schedules robust against frame loss is the most common kind of reliability in the TSN scheduling literature. Park et al. [77] handle frame loss by allowing the retransmission of frames. Schedules for such a scenario must schedule enough time between frame transmissions such that retransmissions do not interfere with other frames. Another way to deal with frame loss is proposed by Feng et al. [70], [74]. In contrast to [77], they do not use retransmissions, but they schedule redundant copies of the same stream over the same path. Zhou et al. [114] approximate the probability of frame loss in a joint routing and scheduling model. Redundant copies of streams are routed over not necessarily disjoint paths to reduce the probability of frame loss. Robustness against permanent single link failures are also covered in several works. There are two approaches in the TSN scheduling literature to handles such failures. First, redundant copies of streams are scheduled and routed over link-disjoint paths before a link failure arises. Examples for such works are [33], [71], [73], and [113]. Huang et al. [68] and Syed et al. [115] use Frame Replication and Elimination for Reliability [17] to implement this approach. Second, streams can be rescheduled and rerouted after a link failure occurred. Pozo et al. [112] present a heuristic for fast rescheduling and rerouting in this case. We remark that all works about computing schedules robust against permanent link failures are also robust against frame loss. Both countermeasures against link failures are also effective against frame loss. Another kind of reliability is considered in [69]. The authors compute schedules robust against clock drift, i.e., clocks of different devices running not with the same speed. They introduce gaps between frame transmissions such that the maximum possible clock drift does not affect other frame transmissions. Unknown hardware bugs or deviations from TSN standards are treated by [118]. The proposed algorithm selects expensive bridges with higher certification for paths

of streams with higher safety requirements. In contrast to all other mentioned works with reliability, Syed et al. [117] use an encoding scheme to reconstruct lost frames. The XORed data of two frames is transmitted over a path disjoint to the paths of both frames.

6) MULTICAST

Every algorithm in literature can be used for multicast streams, as a multicast stream can be substituted by a set of unicast streams. However, the number of streams negatively affects the solving time for a problem instance. Tables 2 and 3 indicate multicast support only for works which include some considerations for the efficient integration of multicast streams without introducing a set of new streams. Most such works handle multicast streams by scheduling only a single frame per link, regardless of the number of consecutive links in the multicast tree of the respective stream. Examples for such works are [31], [41], [59], [83], [85], and [128]. An analysis of the joint routing and scheduling problem with multicast streams is presented in [132]. Yu et al. [136] compute routings and schedules for multicast streams such that migrating a virtual machine sending or receiving TT streams can be done easily. Some works allow multicast streams, but do not elaborate on the implementation details, e.g., [22].

7) TASK SCHEDULING

Only a few research works are concerned with the joint scheduling of streams and tasks. Some of them have integrated dependencies between streams and tasks, i.e., tasks can only be scheduled after some stream has arrived. Such works are presented in [83], [84], [85], [87], [88], and [104]. Other works focus on the scheduling of tasks which produce TT streams while considering safety and security considerations. Preliminary results for this scenario are presented in [72] and extended in [55].

8) SECURITY CONSIDERATIONS

While many works focus on the reliability of data transmissions, security aspects were mostly ignored so far. An exception to this are the works of Reusch et al. [55], [72]. The authors identified the problem of replay and impersonation attacks. However, security aspects are not covered by current TSN standards. The authors propose to use the TESLA protocol [146] to mitigate these problems. The additional messages for key exchanges and the additional tasks for verification and key management are considered during scheduling.

B. SCHEDULING OBJECTIVE

Objective functions are used to measure the quality of solutions and to compare them. We discuss common objectives from the literature and classify research works by their objective. Table 5 shows which research work features which kind of objective.

TABLE 5. Categorization of research works based on optimization objectives.

Objective Category	Research work
No objective	Alnajim *et al.* [32], Ansah *et al.* [45], Atallah *et al.* [113], Atallah *et al.* [71], Bujosa *et al.* [54], Dai *et al.* [82], Dobrin *et al.* [75], Falk *et al.* [96][108], Farzaneh *et al.* [93], Gavriluţ *et al.* [128], He *et al.* [111], Kim *et al.* [42], Li *et al.* [130], Li *et al.* [81], Mahfouzi *et al.* [95], Mahfouzi *et al.* [140], Raagaard *et al.* [76], Santos *et al.* [59], Steiner [7], Steiner *et al.* [41], Syed *et al.* [135] [33] [117], Vlk *et al.* [48], Wang *et al.* [78], Wang *et al.* [46], Xu *et al.* [99], Yao *et al.* [66], Zhou *et al.* [63], Zhou *et al.* [114], Zhou *et al.* [121]
Latency and Jitter	Arestova *et al.* [103] [88], Barzegaran *et al.* [85], Barzegaran *et al.* [84], [100], Dürr *et al.* [28], Feng *et al.* [70], Gärtner *et al.* [79][80], Hellmanns *et al.* [40], Houtan *et al.* [62], Huang *et al.* [98], Huang *et al.* [68], Kim *et al.* [43][44], McLean *et al.* [87], Nie *et al.* [97], Oliver *et al.* [31], Pahlevan *et al.* [104], Pang *et al.* [77], Pei *et al.* [65], Schweissguth *et al.* [8] [132], Vlk *et al.* [51], Wang *et al.* [50], Yang *et al.* [131], Zhang *et al.* [22], Zhou *et al.* [141]
Queuing	Craciunas *et al.* [29], Feng *et al.* [74], Gavriluţ *et al.* [125], Pop *et al.* [57], Vlk *et al.* [51], Vlk *et al.* [109]
Other Traffic	Barzegaran *et al.* [64], Berisa *et al.* [129], Gavriluţ *et al.* [125] [123], Houtan *et al.* [62], Reusch *et al.* [52], Smirnov *et al.* [102], Wang *et al.* [67]
Routing	Li *et al.* [122], Li *et al.* [138], Li *et al.* [133], Schweissguth *et al.* [132], Wang *et al.* [107], Yu *et al.* [134], Yu *et al.* [136]
Topology Synthesis	Gavriluţ *et al.* [73], Reusch *et al.* [55], Xu *et al.* [101], Zhou *et al.* [118]
Reliability	Craciunas *et al.* [69], Park *et al.* [90], Pozo *et al.* [112]
GCL Synthesis	Kentis *et al.* [106], Jin *et al.* [30]
Other	Bhattacharjee *et al.* [145], Chaine *et al.* [53], Feng *et al.* [83], Ginthör *et al.* [91], Gong *et al.* [143], Jin *et al.* [89], Lin *et al.* [92], Min *et al.* [47], Syed *et al.* [116], Syed *et al.* [115], Yang *et al.* [142], Yang *et al.* [142]

1) NO OBJECTIVE

Many research works have no scheduling objective and only try to find some schedule which fulfills all constraints, e.g., [76], [96], [108], and [111]. We note that many SMT approaches feature no objective [7], [41], [59], [93], [95], [140]. In contrast to ILP solving, SMT solvers were not originally designed for optimization. Therefore, many SMT approaches focus on finding a feasible schedule.

2) LATENCY AND JITTER

TSN and the TAS were designed for traffic with hard real-time requirements. Therefore, latency and jitter of streams are interesting properties of schedules. Objective functions including them are the most common kind of objectives in TSN schedule optimization. Oliver et al. [31] and Barzegaran et al. [84] minimize the per-stream jitter. Minimizing the flowspan, i.e., the time all TT stream arrive

at their destination, is a common objective. Examples of approaches using this objective include [28], [40], [50], [62], [68], [100], [103], and [104]. A related but different objective is the minimization of end-to-end delays of TT streams [8], [51], [65], [77], [79], [131], [132]. Kim et al. [43], [44] minimize multiple objectives weighted by constant factors. They take end-to-end delays, jitter, and bandwidth occupation into account. Barzegaran et al. [85] use a combination of jitter and end-to-end latency as measure of schedule quality. Nie et al. [97] minimize end-to-end latency and transmission offsets simultaneously. We remark that these objectives are not competing, as opposed to most multi-criterion problems. Minimization of transmission offsets is also pursued by [22] and [98] which is related but not equal to flowspan or end-to-end latency minimization. Zhou et al. [141] use a combination of jitter, end-to-end delays, number of scheduled streams, and link utilization.

3) QUEUING

Research works which apply isolation constraints for queuing often use more than one queue per egress port for scheduled traffic. In that way, they are able to schedule more streams as isolation only concern streams in the same egress queue. The assignment of streams to egress queues per egress port is a degree of freedom in the respective scheduling problems. Therefore, they try to minimize the number of queues reserved for TT streams per egress port, as the remaining queues are available for other traffic. Examples for such works include [29], [51], [57], [74], [109], and [125].

4) OTHER TRAFFIC

The schedule of TT streams has an influence on the Quality of Service for other traffic classes. Current approaches for scheduling in TSN focus on AVB traffic and BE traffic. For the joint scheduling of TT and AVB streams, Gavriluţ et al. [123], [125] minimize the tardiness of AVB streams as their deadlines are considered to be not strict. Another objective related to AVB streams is used in [129]. The presented heuristic has the objective to schedule as many AVB streams as possible. Yang et al. [131] minimize the the weighted sum of stream latencies of scheduled traffic, AVB, and BE streams. Wand et al. [67] use CQF to shape AVB streams in their approach. The objective function aims to load balance the AVB frame transmissions between the time slots of the CQF mechanism. This reduces the probability that non-periodic BE traffic overloads such a slot. The authors of [52] minimize the occupation percentage of egress ports, i.e., the percentage of a hyperperiod with no active transmission window for TT traffic. The rational of this is that low occupation corresponds to long and frequent time intervals available to other traffic. A similar objective is used in [64] as the authors minimize the average bandwidth occupied by transmission windows for scheduled traffic. Smirnov et al. [102] use a multi-criterion objective for joint routing and scheduling. They reduce the influence of scheduled traffic to other traffic, and simultaneously minimize the number of GCL entries needed to deploy a schedule. The work in [62] focuses on comparing the influence of different objective functions to the QoS of BE traffic. They propose minimization and maximization of frame offsets, hoping that grouping frames together increases the QoS. Additionally, they also suggest two objectives which maximize the gaps between consecutive frame transmissions on a link. They assume that starvation of other traffic classes is reduced in this way.

5) ROUTING

Research works about joint routing and scheduling often consider the quality of the routing in their objective. All of them have in common that the length of the paths is minimized. This is reasonable as longer paths correspond to higher link utilizations, end-to-end latencies, and harder scheduling instances. Schweissguth et al. [132] propose a multi-objective optimization for joint routing and scheduling. First, routing and schedule with minimized path lengths are computed. The obtained path lengths are used as maximum path lengths per stream in a second run. The second run minimizes end-to-end latencies. The joint routing approach of [107] minimizes the number of links in the routing. Li et al. [138] simultaneously minimize the path lengths and a measure for scheduling conflicts of streams routed over the same link. Yu et al. [134] schedule and route streams one after another. They minimize a weighted sum of the number of links used for the currently scheduled stream, and the bandwidth utilization. Li et al. [133] simultaneously minimize path lengths in the routing, and the flowspan. Yu et al. [136] consider the migration of sources of TT streams. They minimize the maximum distance from all possible source nodes of a stream to all destination nodes in a multicast tree. Li et al. [122] maximize the number of streams which are scheduled and routed, and also try to minimize the maximum link load as a secondary objective.

6) TOPOLOGY SYNTHESIS

In addition to joint routing and scheduling, some works also construct the network topology. TSN bridges are expensive, and thus such objectives always include costs for bridges. Gavriluţ et al. [73] minimize multiple objectives weighted by constant factors. The first objective is the tardiness of TT streams to guide their GRASP heuristic to solutions with no deadline misses. The second objective is topology costs. A similar objective is used in [55]. The weighted sum of routing and schedule costs is minimized. Routing costs constitute of overlap penalties for redundant paths and path lengths. Schedule costs constitute of punishments for not schedulable streams and stream latencies. Another approach which minimizes topology costs is proposed in [118]. Selecting bridges from a library is part of the presented problem, which imposes costs for bridges and additional costs when multiple vendors are used. Xu et al. [101] minimize the number of bridges needed to schedule and route all streams such that the utilization is maximized.

7) RELIABILITY

Reliability requirements can be ensured by constraining the set of feasible solutions. However, some works choose to maximize reliability for their respective fault model. Pozo et al. [112] maximize the idle times of links and frames, as such schedules are easier to repair upon link failure. Craciunas et al. [69] maximize the allowed out-of-sync clock drift to cope with synchronization problems and maximize robustness against clock drift. Park et. al. [90] maximize the number of times a frame can be retransmitted without missing its deadline, as they include preemption in their model.

8) GCL SYNTHESIS

TSN bridges do not have an unlimited number of GCL entries per egress port. The minimization of GCL entries is considered by [30]. The reason for this is that the authors propose an incremental approach and the overall number of needed GCL entries is not known in advance. Reducing the number of gate events also reduces the number of guard bands which is beneficial for BE traffic. Kentis et al. [106] minimize the GCL schedule duration. However, it is not clear why schedule duration matters, as the limiting factor in TSN hardware is the number of GCL entries.

9) OTHERS

Some research works use a problem specific objective not comparable to other works. We present them for the sake of completeness. The authors of [89] minimize the number of frames as they propose a joint approach for scheduling and message fragmentation. Syed et al. [115] and [116] use a modelling specific objective which is related to load balancing of ports in an in-vehicle architecture with one central processing unit. Ginthür et al. [91] minimize the wasted bandwidth for different link layer technologies, i.e., Ethernet and 5G links. Feng et al. [83] minimize the response time of tasks which may be dependent on streams as they consider the joint scheduling of streams and tasks. Chaine et al. [110] maximize the length of transmission time windows of streams at their respective talkers such that latency and jitter requirements are met. This is the only work in TSN scheduling which employs a quadratic objective function. Bhattacharjee et al. [145] employ a multi-criterion objective. Their first objective is to minimize the maximum load across all servers as the considered problem includes the placement of talker applications. The second objective is to minimize the average hop count of all streams. Lin et al. [92] present a heuristic for incremental scheduling that aims to maximize the probability that more streams can be added later. This is required in industrial use cases as turning off machines to deploy a new schedule may be expensive. Yang et al. [142] state that they maximize the number of scheduled streams while minimizing the link occupancy rate. However, this rate is not defined in the published magazine article. Gong et al. [143] also maximize the occupancy rate while minimizing the maximum link utilization. They define the occupancy rate as the fraction of bandwidth reserved for scheduled traffic actually used for transmissions. Min et al. [47] compare metaheuristics by maximizing the number of scheduled stream for the same problem instance.

TABLE 6. Overview of investigated problem instances in the literature of the scheduling problem with fixed routing.

Research work	Topology	ES	Bridges	TT streams
Arestova *et al.* [88]	Mesh, ring	N/A	5 – 30	N/A
Barzegaran *et al.* [85]	Various	6 – 20	2 – 20	8 – 27
Barzegaran *et al.* [64]	Various	3 – 31	2 – 15	7 – 137
Barzegaran *et al.* [84]	Various	5 – 20	2 – 20	8 – 27
Bujosa *et al.* [54]	Line	3 – 9	1 – 3	N/A
Chaine *et al.* [53]	Line, spacecraft	2 – 31	4 – 15	15 – 304
Craciunas *et al.* [29]	N/A	3–7	1–5	5–1000
Craciunas *et al.* [69]	Tree	16	7	96
Dai *et al.* [82]	Link	N/A	N/A	3
Dürr *et al.* [28]	ER, RRG, BA	24–100	5–20	30–1500
Farzaneh *et al.* [93]	Snowflake	12	2	14–100
Feng *et al.* [70]	N/A	2 – 8	2 – 5	2 – 16
Feng *et al.* [74]	N/A	5 – 8	3 – 5	N/A
Feng *et al.* [83]	N/A	50	10	11
Gärtner *et al.* [79][80]	Line, machine	2 – 54	2 – 15	1 – 104
Gavriluţ *et al.* [125]	Star, Snowflake	3–32	1–18	4–35
Gavriluţ *et al.* [128]	Various	7–31	1–15	20–186
Ginthör *et al.* [91]	N/A	N/A	N/A	10
Hellmanns *et al.* [40]	Ring of rings	100–2500		10–2000
Hellmanns *et al.* [144]	Ring of rings	N/A	N/A	50–500
Houtan *et al.* [62]	Snowflake	6	2	10
Huang *et al.* [68]	Automotive	33	14	20 – 480
Jin *et al.* [89]	N/A	5–30	5–30	10–60
Jin *et al.* [30]	N/A	N/A	6–20	10–10000
Kim *et al.* [42]	Various	7	5	6
Kim *et al.* [43][44]	Automotive	17	4	27
Li *et al.* [81]	Various	30 – 100	7 – 20	3 – 100
Lin *et al.* [92]	Tree, mesh, ring	4 – 16	1 – 4	~ 48 – 223
McLean *et al.* [87]	Tree, mesh, ring	4 – 432	2 – 43	16 – 1728
Min *et al.* [47]	Real-world	14		300
Oliver *et al.* [31]	Line	50	10	10–50
Pang *et al.* [77]	In-train, spacecraft	31 – 54	13 – 31	N/A
Park *et al.* [90]	Automotive	5 – 20	3 – 7	100 – 500
Pei *et al.* [65]	Spacecraft	31	15	20
Pop *et al.* [57]	N/A	3–5	1–2	3–5
Raagaard *et al.* [76]	N/A	4–402		15–290
Reusch *et al.* [72]	Various	4–32	1–16	N/A
Reusch *et al.* [52]	Snowflake	3–256	1–146	14–316
Santos *et al.* [59]	N/A	50	10	1 – 10
Steiner [7]	Star, tree, snowflake	N/A	N/A	100 – 1000
Steiner *et al.* [41]	N/A	50	10	10–50
Vlk *et al.* [48]	Ring of lines, ring of trees, spacecraft	~ 20 – 1700	~ 20 – 300	1500 – 12000
Vlk *et al.* [51]	Mesh, ring, tree	4 – 16	1 – 4	6 – 450
Wang *et al.* [50]	N/A	4 – 10	3 – 9	10 – 100
Wang *et al.* [46]	Automotive	4	1	3
Wang *et al.* [78]	Mesh	15 – 16	6 – 8	N/A
Wang *et al.* [67]	Line, ring, tree	N/A	5 – 200	N/A
Yao *et al.* [66]	N/A	N/A	N/A	10 – 20
Zhou *et al.* [63]	Automotive	5	16	N/A
Zhang *et al.* [22]	Tree, line, tree	6 – 8		200 – 1200

TABLE 7. Overview of investigated problem instances in the literature of the joint routing problem.

Research work	Topology	ES	Bridges	TT streams
Alnajim *et al.* [32]	N/A	30–150	10–50	100–1500
Arestova *et al.* [103]	N/A	50	10	10 – 100
Atallah *et al.* [113]	N/A	6 – 24	N/A	30 – 600
Atallah *et al.* [71]	N/A	N/A	3–21	20–60
Berisa *et al.* [129]	Ring, full mesh, spacecraft	13 – 31	4 – 15	222
Bhattacharjee *et al.* [145]	Ring, BA, ER	46 – 1024	24 – 512	90 – 1845
Falk *et al.* [96]	Line, ring, BA, ER	5–36		2–30
Falk *et al.* [108]	Ring w/ k neighbors	50–400		50–150
Gavriluţ *et al.* [73]	N/A	4–20	N/A	4–38
Gavriluţ *et al.* [123]	Various	3–256	2–146	4–427
Gong *et al.* [143]	N/A	N/A	N/A	15 – 180
He *et al.* [111]	ER, RRG, BA	20		25 – 200
Huang *et al.* [98]	Spacecraft, ER	N/A	N/A	500 – 2000
Kentis *et al.* [106]	Ring, mesh	12	12	20–52
Li *et al.* [122]	ER, RRG, BA	10	10	10 – 100
Li *et al.* [130]	Mesh, automotive	15 – 105	3 – 21	2 – 4000
Li *et al.* [138]	Real-world	31	13	100 – 300
Li *et al.* [133]	N/A	39	16	10 – 40
Mahfouzi *et al.* [95][140]	ER, Automotive	20	10–45	19–106
Nie *et al.* [97]	Ring, mesh	11 – 14	4 – 15	10 – 40
Pahlevan *et al.* [104]	Grid, ring	27–45	9	30–40
Pahlevan *et al.* [100]	Grid, ring	50	10	60–100
Pozo *et al.* [112]	Synthetic	6–8	3–8	10–50
Reusch *et al.* [55]	Various	4 – 128	2–64	2 – 144
Reusch *et al.* [72]	Various	4–32	1–16	N/A
Schweissguth *et al.* [8]	Ring, mesh	13	N/A	60
Schweissguth *et al.* [132]	Ring, mesh	12	12	25 – 40
Smirnov *et al.* [102]	N/A	N/A	N/A	5 – 75
Syed *et al.* [116]	Automotive	N/A	N/A	20–90
Syed *et al.* [135]	Automotive	N/A	N/A	100–500
Syed *et al.* [33]	Automotive	N/A	N/A	100–500
Syed *et al.* [115]	Automotive	N/A	N/A	20–90
Syed *et al.* [117]	Automotive	N/A	N/A	112
Vlk *et al.* [109]	Ring, mesh	12 – 48	12 – 48	10 – 300
Wang *et al.* [107]	N/A	25	5	10–100
Xu *et al.* [101]	Mesh	5	1 – 23	4 – 12
Xu *et al.* [99]	Mesh, ring	N/A	7–15	40 – 80
Yang *et al.* [142]	N/A	2	N/A	45 – 180
Yang *et al.* [131]	Star, tree, BA	N/A	8 – 18	150 – 900
Yu *et al.* [134]	Mesh	72	24	1 – 500
Yu *et al.* [136]	BA	72	24	10 – 350
Zhou *et al.* [118]	ER, automotive	N/A	6–10	50 – 240
Zhou *et al.* [114]	ER, automotive	16	6–10	50–380
Zhou *et al.* [121]	ER, real-world	16	4–64	30 – 220
Zhou *et al.* [141]	Mesh	≥ 4	3 – 15	50 – 650

C. PROBLEM INSTANCES

Before we describe evaluation results, we describe the problem instances used for evaluations in the literature. We present an overview of used network topologies, network sizes, and numbers of streams. Tables 6 and 7 compile this information about the problem instances used for evaluations with fixed routing and joint routing, respectively.

Unfortunately, some research works do not elaborate on the used topologies, which makes assessing and comparing the results to other works harder. Most research works only use synthetic test cases. Ring topologies are commonly used in evaluations, e.g., in [8], [40], [67], [87], [88], [96], [97], [100], [101], [104], [108], [109], and [132]. Hellmanns et al. [40] argue that rings are a common topology in real-world industrial facilities. Other systematic topologies used include line [31], [67], [96], grid [100], [104], and snowflake-like [52], [62], [93], [125] networks. Various randomly generated topologies are also used in evaluations. Erdós-Rényi graphs (ER) are the most common ones [28], [96], [111], [114] [118], [121], [122], but Barabási-Albert graphs (BA) [28], [96], [111], [122] [134] and random regular graphs (RRG) [28], [111], [122] are also used. A few research works features evaluations with real-world topologies. Syed et al. [33], [115], [116], [117], [135] use a real-world automotive architecture for their evaluations. A large automotive architecture including stream parameters is discussed in [68]. Other automotive architectures are used by Kim et al. [43], [44], Li et al. [130], Mahfouzi et al. [95], [140], and Wang et al. [46]. Zhou et al. [114], [118] conclude their evaluations by investigating a real-world example from General Motors. The authors of [73] also use a real-world problem instance from General Motors. However, this instance is only a set of streams without topology. Min et al. [47] use the network topology of the National Science Foundation of the United States of America. Barzegaran et al. [64] presents evaluations with real-world test cases from General Motors and a real-world spacecraft. Pang et al. [77] evaluate an algorithm for schedule updates in a real-world in-train network and a spacecraft. Similarly, the authors of [79] evaluate their algorithm for schedule reconfigurations with the topology of a not specified machine. Vlk et al. [48], Chaine et al. [53], Huang et al. [98], Gavriluţ et al. [128], and Berisa et al. [129], perform evaluations with a real-world spacecraft topology.

All research works concerned with synthetic test cases use randomly generated streams. Sources and destinations of these streams are selected uniformly from the sets of talkers and listeners in the respective topology. The number of streams varies considerably between different research works. It ranges from 2 streams in the smallest instance of Falk et al. [96] to up to 10812 streams in the largest instance of Vlk et al. [48]. All works, which describe the placement of deadlines, place them at the end of the respective stream's period. No research work allows deadlines to be after the end of the hyperperiod a frame was sent. Stream periods range from 32 μs in [30] to 500 ms in [29]. All research works assume transmission rates of either 100 Mb/s or 1 Gb/s per egress port.

D. SCALABILITY

Scheduling in TSN is known to be NP-complete. Therefore, solving times and sizes of feasible problem instances matter. Almost all research works about TSN scheduling include

or even focus on evaluating the scalability of the respective proposed approach. These evaluations measure the solving times for selected problem instances. Tables 8 and 9 compile the reported runtimes needed to solve the largest problem instance for which a schedule was found in the respective research work. Tables 10 and 11 report the same results for research works which feature the joint computation of schedules and routings. We divided results in separate tables for exact and heuristic algorithms for better comparability. In cases where it was not clear which problem instance can be considered as the largest one, we used the number of streams as tie-breaker. This is justified by several research works surveyed in this paper, e.g., in [96]. The tables are meant to show general tendencies and improvements, not to suggest one approach over the other. Caution is needed when interpreting the tables. It shows the reported times after which an algorithm terminated, not the time until a first valid schedule was obtained, as almost all papers do not report this time. This is a systematic disadvantage of exact approaches as they only terminate when the optimal solution is found or some timeout is reached, while heuristic algorithms may terminate much earlier with suboptimal solutions. Some research works deal with more parameters than the size of the network and the number of streams, e.g., Oliver et al. [31] present evaluations about the influence of the number of transmission windows per egress port to scalability. Other works handle problem extensions, e.g., AVB or task scheduling. Approximations are given when results are not stated in the text and had to be estimated by the presented figures. Ranges are given when multiple instances are considered to be the largest. We identified two tendencies with respect to solving times.

First, heuristic approaches can handle larger instances than approaches with exact solution methods. While the number of network nodes is approximately in the same range, heuristic algorithms can schedule problem instances with more streams compared to exact approaches. Typical numbers of streams in exact approaches are less than 100, e.g., in [8], [69], and [96]. However, there are some notable exceptions. Craciunas et al. [29] present an incremental scheduling algorithm with backtracking, which scheduled instances with 1000 streams in their evaluations. Later works present incremental approaches which were able to schedule as many as 2000 streams [98]. Oliver et al. [31] assigned streams to transmission windows of egress ports and report solved instances with 750 streams. Heuristic approaches were able to schedule instances with more than 10000 streams, e.g., [48] and [30].

Second, exact approaches which solve the joint routing and scheduling problem can only handle instances with smaller numbers of nodes compared to approaches solely for scheduling. Typical networks in evaluations of joint routing algorithms contain less than 50 nodes [71], [72], [73]. This is due to the solution space growing heavily with an increased number of possible paths per stream. However, there are approaches able to compute routings and schedules for problem instances with up to 96 nodes [109], [134]. Most networks in the literature of scheduling with a fixed routing contain less than 96 nodes. The range of the number of

TABLE 8. Overview of solving times of the respective largest reported problem instance for which a schedule was found. Only research works with exact approach and fixed routing are included for comparability.

Research work	Solution approach	Nodes	TT streams	Runtime (s)
Barzegaran *et al.* [64]	CP	120	500	N/A
Barzegaran *et al.* [84]	CP	40	27	2563
Chaine *et al.* [53]	ILP	46	304	~ 240
Craciunas *et al.* [29]	SMT	12	1000	< 18000
Craciunas *et al.* [69]	SMT	23	96	~ 0.343 – 0.437
Dai *et al.* [82]	CP	N/A	3	N/A
Farzaneh *et al.* [93]	SMT	14	100	< 240
Feng *et al.* [70]	SMT	13	16	N/A
Feng *et al.* [74]	SMT	13	N/A	~ 900
Feng *et al.* [83]	SMT	60	11	384 – 694
Ginthör *et al.* [91]	CP	N/A	10	N/A
Houtan *et al.* [62]	SMT	8	10	0.37– 1153.52
Jin *et al.* [89]	SMT	4	4	< 600
Jin *et al.* [30]	SMT	6	50	660–1080
Li *et al.* [81]	SMT	120	100	~ 320 – 450
Nie *et al.* [97]	ILP	26	40	~ 0 – 10
Oliver *et al.* [31]	SMT	20	750	~ 600
Pang *et al.* [77]	ILP	86	N/A	~ 620 – 900
Pop *et al.* [57]	ILP	7	5	80.19
Santos *et al.* [59]	SMT	60	10	< 288000
Steiner [7]	SMT	N/A	1000	~ 180 – 1140
Steiner *et al.* [41]	SMT	60	50	~ 0.01 – 100
Vlk *et al.* [51]	ILP	20	414	≤ 600
Zhou *et al.* [63]	SMT	21	N/A	N/A

TABLE 9. Overview of solving times of the respective largest reported problem instance for which a schedule was found. Only research works with heuristic approach and fixed routing are included for comparability.

Research work	Solution approach	Nodes	TT streams	Runtime (s)
Arestova *et al.* [88]	Heuristic	> 30	N/A	105.4
Atallah *et al.* [113]	Heuristic	24	600	~ 8
Barzegaran *et al.* [85]	CP + heuristic	40	27	3553 – 9153
Barzegaran *et al.* [84]	CP + heuristic	40	27	161
Bujosa *et al.* [54]	Heuristic	12	N/A	< 0.01
Dürr *et al.* [28]	Tabu search	120	1500	11520
Gärtner *et al.* [79][80]	Heuristic	69	104	~ 0.1
Gavriluţ *et al.* [125]	GRASP	50	35	612.8
Gavriluţ *et al.* [128]	GRASP	46	186	750
Hellmanns *et al.* [40]	Tabu search	2500	2000	~ 4400
Huang *et al.* [68]	Heuristic	47	480	~ 2700
Jin *et al.* [30]	Heuristic	20	10000	~ 5100
Kim *et al.* [42]	Heuristic	12	6	N/A
Kim *et al.* [43][44]	GA	21	27	12300
Lin *et al.* [92]	Heuristic	20	~ 223	N/A
McLean *et al.* [87]	Heuristic	475	1728	~ 10000
Park *et al.* [90]	GA	< 27	500	N/A
Pei *et al.* [65]	Heuristic	46	20	N/A
Raagaard *et al.* [76]	Heuristic	402	290	20–54
Reusch *et al.* [52]	Heuristic	402	316	10.52
Vlk *et al.* [48]	Heuristic	2000	10812	~ 1000
Wang *et al.* [50]	Machine learning	19	100	~ 400
Wang *et al.* [46]	Heuristic	5	3	< 1
Wang *et al.* [78]	Heuristic	23	N/A	N/A
Wang *et al.* [67]	Heuristic	16	200	~ 1000
Yao *et al.* [66]	Heuristic	N/A	20	~ 2
Zhang *et al.* [22]	Heuristic	8	1200	~ 11

TABLE 10. Overview of solving times of the respective largest reported problem instance for which a schedule was found. Only research works with exact approach and joint routing are included for comparability.

Research work	Solution Approach	Nodes	TT streams	Runtime (s)
Atallah *et al.* [71]	ILP	14	60	~ 100
Falk *et al.* [96]	ILP	8	30	~ 170 – 1580
Gavriluţ *et al.* [73]	CP	20	38	172800
Hellmanns *et al.* [144]	ILP	N/A	N/A	50–500
Huang *et al.* [98]	ILP	44	2000	1620
Li *et al.* [133]	ILP	55	40	~ 80
Mahfouzi *et al.* [95][140]	SMT	65	45	~ 21
Pozo *et al.* [112]	ILP	16	50	2–85
Reusch *et al.* [55]	CP	48	33	1500
Reusch *et al.* [72]	CP	48	N/A	~ 1800
Schweissguth *et al.* [8]	ILP	24	52	1284.53
Schweissguth *et al.* [132]	ILP	24	~ 46 – 60	N/A
Smirnov *et al.* [102]	PBO	N/A	75	~ 42 – 78
Syed *et al.* [116]	ILP	N/A	90	~ 21000
Vlk *et al.* [109]	CP	96	300	~ 100
Xu *et al.* [101]	SMT	24	12	~ 144000
Xu *et al.* [99]	SMT	12	80	~ 850
Yu *et al.* [134]	ILP	96	450	N/A
Zhou *et al.* [118]	SMT	10	240	~ 3500 – 4000
Zhou *et al.* [114]	SMT	10	380	~ 2700
Zhou *et al.* [121]	SMT	24	220	~ 55 – 440

streams is approximately the same for approaches with fixed routing and joint routing.

VI. PUBLICATION HISTORY

We give an overview of the publication history of TSN scheduling. First, we highlight seminal works from the literature. Then, we analyze the development of the field with respect to the number of published papers per year.

A. SEMINAL WORKS

Early works about per-flow scheduling in Ethernet networks were presented by Steiner [7] and Schweissguth et al. [8]. While these works are not specifically for TSN and abstract on the details of the real-time enhancement for Ethernet. they influenced many later works presented in this survey. The first works specifically about scheduling in TSN were presented in 2016. Dürr et al. [28] presented an ILP for no-wait scheduling and identified the problem of guard bands consuming bandwidth. Craciunas et al. [29] adapted the work of Steiner [7] for TSN. They introduced isolation constraints and incremental scheduling to the domain of TSN. Gavrilut et al. [73] is the first work which features joint routing and reliability considerations. Raagard et al. [76] introduced reconfiguration of schedules to TSN scheduling. Oliver et al. [31] proposed a scheduling approach with limits the number of used GCL entries by computing them in a joint approach with transmission offsets. All earlier works computed GCLs by a post-processing after scheduling.

B. PUBLISHED PAPERS

Figure 19 shows the number of papers about TSN scheduling per year. The first papers about scheduling in TSN were published in 2016. The general trend is that the field grows almost monotonously from one year to the next, with only one exception in 2019. We observe a significant increase in published works since the year 2020. Given the fast growth of the last 2 years, we expect even more research works about TSN scheduling in the future.

TABLE 11. Overview of solving times of the respective largest reported problem instance for which a schedule was found. Only research works with heuristic approach and joint routing are included for comparability.

Research work	Solution Approach	Nodes	TT streams	Runtime (s)
Alnajim *et al.* [32]	Heuristic	200	1500	2718
Arestova *et al.* [103]	Genetic algorithm	60	100	~ 470
Berisa *et al.* [129]	Heuristic	46	222	602 – 7090
Bhattacharjee *et al.* [145]	SA	1536	1845	~ 1800
Falk *et al.* [108]	Heuristic	400	400	~ 6000 – 11000
Gavriluţ *et al.* [123]	GRASP	402	427	534.6
Gavriluţ *et al.* [73]	GRASP	20	38	558
	Heuristic	20	38	130
Gong *et al.* [143]	Tabu search	N/A	180	~ 11160
He *et al.* [111]	Machine learning	20	200	7
Kentis *et al.* [106]	Heuristic	≥ 13	60	N/A
Li *et al.* [122]	Heuristic	20	100	N/A
Li *et al.* [130]	Heuristic	126	4000	0.437 – 0.88
Li *et al.* [138]	ILP + heuristic	44	300	~ 0.3
Pahlevan *et al.* [104]	List scheduler	45	40	0.014
	Genetic algorithm	45	40	56.75
Pahlevan *et al.* [100]	List scheduler	50	100	0.103
	Heuristic	50	100	1.58
Reusch *et al.* [55]	SA + list scheduler	192	144	1200
Syed *et al.* [33]	Heuristic	N/A	500	0.41
Syed *et al.* [135]	Heuristic	N/A	500	0.170
Syed *et al.* [115]	Heuristic	N/A	90	~ 342
Syed *et al.* [117]	Heuristic	N/A	90	~ 8 – 10
Wang *et al.* [107]	Heuristic	30	100	72
Yang *et al.* [142]	Tabu search	N/A	180	~ 11160
Yang *et al.* [131]	Machine learning	18	900	N/A
Yu *et al.* [136]	Heuristic	96	350	N/A
Zhou *et al.* [141]	Heuristic	N/A	650	~ 2700

VII. FUTURE WORKS

In this section, we discuss the results of the literature study. First, we suggest improvements for future research works. Then, we highlight open problems not handled sufficiently so far.

A. SUGGESTIONS FOR IMPROVEMENT

The surveyed literature features many high quality research works. However, there is room for improvement in the presentation of some of these works. We suggest improvements in the hope that the overall quality of the TSN scheduling literature can be improved even more in the future. First, we discuss shortcomings and improvements in the presentation of evaluation methodologies. Then, we suggest the use of the technical terminology used in Ethernet bridging.

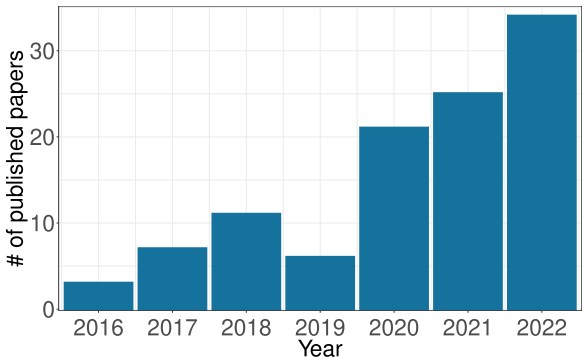


FIGURE 19. Number of published research works per year about TSN scheduling. Only works published before March 2023 are counted.

1) EVALUATION METHODOLOGIES

The scalability of the proposed solutions from the literature was extensively evaluated. Unfortunately, the impact of possible additional features and changes in parameters to the various objective functions was mostly ignored so far. Although scalability is an important property of a scheduling algorithm, it would be interesting to see more evaluations regarding solution quality. Most test cases in the literature are synthetically constructed, both network topology and streams. Even though it is hard to obtain test cases from the industry, let alone publish them, we would like to see more evaluations with realistic instances. It is not clear whether the proposed algorithms are suitable for large industry-scale instances or how they look like. It is also extremely difficult to compare the results of different research works as there is no public set of test cases for benchmarks. Consequently, there is little research work available about which algorithm should be used in which setting.

Unfortunately, it is also hard to assess the significance of evaluation results in some papers for two reasons. First, the instances solved are not sufficiently described. At least the topology and a description of assumed delays, e.g., processing and propagation delay, should be contained in the description of the network. Important properties of streams like deadlines or periods are often missing. Second, some evaluations report results for individual problem instances and are thus more of anecdotal character. There are easy and hard instances for every algorithm. Comparing multiple approaches on the same selected instances can be useful, but this may have the taste of picking specific instances in support of some conclusion. Instead of reporting results for individual instances, average results for multiple instances with the same evaluation setting should be reported. However, we acknowledge that evaluations for specific scenarios can be of interest. Especially readers from the industry may enjoy a rigorous report about a specific running system. We only want to emphasize that strong conclusions should be backed up by strong evidence and extensive evaluation. Another property covered by many evaluations is the schedulability of the respective proposed approach. These evaluations treat instances as infeasible when no schedule was found before some timeout. Thus, comparing the schedulability of two scheduling approaches which support different features is biased, as timeouts do not prove infeasibility. This may lead to wrong conclusions in favor of some algorithm or model, although schedulability is actually equal.

2) TERMINOLOGY

Many works surveyed in this paper use a vocabulary loosely related to Ethernet bridging. However, the standards and other relevant literature use a specific technical terminology. We suggest that the scheduling community adopts this jargon. Readers from adjacent research domains or who have prior knowledge in Ethernet bridging can benefit from a consistent vocabulary. The word *stream* is used in several standards, e.g., [11], [12], and [147]. Therefore, we suggest to use *stream* instead of *flow*. Network devices which send or receive data streams are denoted as *end stations* in the original bridging standard IEEE 802.1D [148]. The source end station of a stream is denoted as *talker*, while the destination end station is denoted as *listener*. Layer 2 switching devices are denoted as *bridges* instead of *switches* in IEEE 802.1D [148] and in the names of many other standards, e.g., in [9] and [14]. Thus, we suggest to use these terms when describing network topologies.

Frames are the units of data transmission, as TSN is a layer 2 technology, while packets are the units of data transmissions in layer 3 technologies (cf. [149]). Although the meaning of the term *packet* is clear in the context of scheduling, it is technically wrong. Routing is the process of path computation on layer 3. Therefore, the term *path selection* is more appropriate in TSN. However, we note that we used the term *routing* in this survey several times. The reason for this is to ensure consistency with the reviewed literature which solves the so-called joint routing problem.

B. OPEN PROBLEMS

The available literature is comprehensive with regard to solution approaches to the unmodified scheduling problem in TSN. However, there is still a wide field of relevant aspects which are not yet understood.

1) IMPACT OF GUARD BANDS AND GCL ENTRIES

To the best of our knowledge, the impact of guard bands on bandwidth available to lower-priority traffic was not evaluated in the literature. Likewise, the impact of available GCL entries on available bandwidth for lower-priority traffic is not investigated in detail. The evaluations so far suggest that AVB streams benefit from schedules with many holes between TT streams with regard to tardiness. However, such schedules may need more gate closings and thus guard bands, which reduces the available bandwidth. It is not clear how AVB and BE traffic can be simultaneously integrated in a unified approach for the scheduling of TT streams.

2) ROUTING AND MULTICAST

The joint routing and scheduling problem was explored in detail in the literature. All research works about this topic agree that schedulability benefits from joint routing and scheduling. However, solving the joint routing problem is significantly harder compared to scheduling with a given routing. Unfortunately, there is currently no exact and scalable approach known for joint routing. Additionally, it is not understood which properties a routing should have to benefit schedule synthesis and quality. TSN supports multicast streams which are relevant in real use cases. Some of the algorithms presented in research works covered by this survey can handle multicast streams. However, the literature lacks evaluations and insights about the appropriate integration of multicast streams in a schedule.

3) ONLINE RECONFIGURATION

There is also little work about online schedule reconfiguration, though it is important for operation. In some scenarios, e.g., automotive networks, insertion and removal of streams at execution time of the schedule can be important. So far it is not explored exhaustively what properties a schedule should have such that reconfiguration can by computed efficiently. However, there are preliminary works about this topic [79], [116]. Instead of rescheduling all streams, many works about reconfiguration try to resolve conflicts by assigning streams to other traffic classes [76] or paths [136], [138]. Other ways to resolve scheduling conflicts instead of rescheduling all streams may be needed in the future. Unfortunately, there are no reconfiguration algorithms for most of the problem extensions from Section III-E.

4) QUEUING AND HANDLING OF NON-DETERMINISM

An important open problem in TSN is sufficient integration of queuing. Almost all research works use isolation constraints from [29], i.e., they do not allow frames of different streams to reside in the same queue at the same time. However, this is not a requirement of the TAS. Some approaches even separate streams by assigning them to different egress queues during scheduling. The rational of this is to reduce the impact of non-determinism like frame loss. Other attempts to reduce the influence of such causes of non-determinism are not yet explored. The benefits of unrestricted queuing regarding schedulability or solution quality has not yet been evaluated.

Real hardware bridges are subject to non-determinism. There is jitter in processing delays, and clocks are not exactly synchronized in reality. Additionally, frames that are scheduled to arrive approximately at the same time at two ingress ports of the same bridge may cause race conditions, i.e., processing order is not deterministic. All research works covered by this survey assume bridges are perfectly deterministic. Thus, the literature lacks handling of such causes of non-determinism.

5) FILTERING AND POLICING

Per-Stream Filtering and Policing (PSFP) is a standard defined in IEEE 802.1Qci [147] for filtering and policing in TSN. Currently, there are no devices available implementing PSFP. However, filtering and policing could be used to prevent violations of schedules through unexpected packets. Packets not scheduled, delayed frames, and frames larger than expected can be filtered at execution time of a schedule. Thus, PSFP requires configuration of filtering entries that need to be derived from the schedule. A joint approach may be needed as PSFP imposes additional restrictions, e.g., the number of available filtering entries will be limited in bridges.

6) SECURITY ASPECTS

The security of real-time Ethernet networks was mostly ignored so far. All considerations for reliability and safety in TSN assume that no malicious party is involved in the communication. The standards do not cover countermeasures against replay or impersonation attacks. This may be a problem for highly vulnerable use cases of TSN, e.g., factory automation and in-vehicle networks. Future scheduling algorithms may integrate security considerations. For instance, key exchange and management result in additional streams that must be protected from other traffic. However, source authentication and integrity may also be implemented by future TSN standards or application layer protocols. Other security problems may be countered with PSFP, e.g., jamming and Denial-of-Service attacks by malicious end stations.

7) LEGACY DEVICES

Traditional Ethernet is a widespread layer 2 technology for industrial applications. Such applications are typically designed to be used for years or even decades. Thus, integrating legacy devices which are not capable of traffic scheduling or time synchronization will be a major problem in the next years for the deployment of TSN. Future scheduling algorithms may help to integrate such devices. For instance, scheduling algorithms can reserve bandwidth for the communication of these devices. However, new devices, e.g., gateways or proxies, may be needed to fully support the coexistence of legacy devices and scheduled traffic.

8) USE OF TSN MECHANISMS

TSN is not limited to scheduled traffic and the TAS. Other traffic classes may have real-time requirements, but cannot be scheduled as the respective streams are not periodic. Different traffic classes may have different sets of real-time requirements, e.g., demanding bounded jitter instead of bounded latency. TSN features more mechanisms which may be applied to fulfill these requirements, such as Asynchronous Traffic Shaping [150] or Cyclic Queuing and Forwarding [18]. A major open problem in TSN is the coexistence of multiple mechanisms and the assignment of

streams to them. Input may be a set of streams or traffic rates with their descriptors and real-time requirements, and output is their assignment to appropriate TSN mechanisms together with the complete network configuration. This problem goes far beyond the TSN scheduling problem, but may impose additional constraints on the latter. Some requirements can only be fulfilled by scheduling the respective streams and computing GCLs for the TAS. Others may not even know the traffic streams in advance and can be implemented by the CBS or even simpler mechanisms. As even computing GCLs for the TAS is a challenging task for current state-of-the-art scheduling and optimization algorithms, such a comprehensive approach is currently unreachable. Hopefully, future works will move towards such long term goals and enable users to exploit the full potential of TSN.

9) UNDERSTANDING OF THE TSN PROBLEM

So far, scalability analyses have been conducted on special algorithms. However, they do not provide insights in what makes the TSN problem hard. This also pertains to all problem extensions like joint routing and multicast, reliability, robustness, BE or ABE traffic, etc. Moreover, properties of schedules such as tightness or average duration of open periods of the TAS have not yet been investigated. It would be helpful to understand the impact of problem extensions on the structure of schedules in an intuitive way. A better understanding of extensions and their impact on schedule structure may facilitate the development of heuristic algorithms that solve larger instances of the TSN problem with acceptable quality compared to exact approaches.

VIII. CONCLUSION

TSN is a set of standards to enable real-time transmission over switched Ethernet networks. IEEE 802.1Qbv [6] defines traffic scheduling combined with the Time-Aware Shaper (TAS), i.e., transmissions of periodic high-priority streams are scheduled such that packets hardly interfere and that ultra-low latency is achieved. Moreover, the TAS protects scheduled traffic against traffic from other traffic classes. This approach requires the configuration of transmission times for streams at the Talkers (source nodes) as well as the configuration of the TAS on the switches.

In this paper, we first gave an introduction to TSN with focus on traffic scheduling and the TAS. We defined the "TSN scheduling problem" and discussed common extensions such as scheduling with fixed or joint routing, various forms of queuing, support for reliability or lower-priority traffic, or respecting technical restrictions. Some of these extensions lead to optimization problems. We summarized frequently used scheduling and optimization methods to tackle these challenges. Then we reviewed a large body of literature about the TSN scheduling problem and classified it regarding the mentioned extensions. Subsequently, we analyzed and compared the works with respect to modelling assumptions, scheduling objectives, problem instances, and scalability, and pointed out advances. We tracked seminal works and identified popular publication venues for TSN scheduling. We discussed the area by suggesting improvements and pointing out open problems.

This survey serves researchers to identify the current state of the art and open problems in TSN scheduling. The many problem extensions suggest that the construction of an efficient scheduling or optimization algorithm which considers all relevant aspects is infeasible. We expect future work to provide a better understanding of the complexity of the TSN scheduling problem to cover more problem extensions while maintaining scalability.

ACKNOWLEDGMENT

The authors thank Manuel Eppler and Lukas Bechtel for valuable input and stimulating discussions.

The authors alone are responsible for the content of the paper.

REFERENCES

[1] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 88–145, 1st Quart., 2019.

[2] A. Minaeva and Z. Hanzálek, "Survey on periodic scheduling for time-triggered hard real-time systems," *ACM Comput. Surv.*, vol. 54, no. 1, pp. 1–32, Jan. 2022.

[3] Y. Seol, D. Hyeon, J. Min, M. Kim, and J. Paek, "Timely survey of time-sensitive networking: Past and future directions," *IEEE Access*, vol. 9, pp. 142506–142527, 2021.

[4] L. Deng, G. Xie, H. Liu, Y. Han, R. Li, and K. Li, "A survey of real-time Ethernet modeling and design methodologies: From AVB to TSN," *ACM Comput. Surv.*, vol. 55, no. 2, pp. 1–36, Feb. 2023.

[5] V. Gavriluţ, A. Pruski, and M. S. Berger, "Constructive or optimized: An overview of strategies to design networks for time-critical applications," *ACM Comput. Surv.*, vol. 55, no. 3, pp. 1–35, Mar. 2023.

[6] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 25: Enhancements for Scheduled Traffic*, IEEE Standard 802.1Qbv-2015, 2016, pp. 1–57.

[7] W. Steiner, "An evaluation of SMT-based schedule synthesis for time-triggered multi-hop networks," in *Proc. 31st IEEE Real-Time Syst. Symp.*, Nov. 2010, pp. 375–384.

[8] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based joint routing and scheduling for time-triggered networks," in *Proc. 25th Int. Conf. Real-Time Netw. Syst.*, Oct. 2017, pp. 8–17.

[9] *IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications*, IEEE Standard 802.1AS-2020, 2020, pp. 1–421.

[10] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE Standard 1588-2019, 2020, pp. 1–499.

[11] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)*, IEEE Standard 802.1Qat-2010, 2010, pp. 1–119.

[12] *IEEE Standard for Local and Metropolitan Area Networks—Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams*, IEEE Standard 802.1Qav-2009, 2010, pp. C1–72.

[13] *IEEE Standard for Local and Metropolitan Area Networks–Audio Video Bridging (AVB) Systems*, IEEE Standard 802.1BA-2021, 2021, pp. 1–45.

[14] *IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks*, IEEE Standard 802.1Q-2018, 2018, pp. 1–1993.

[15] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 24: Path Control and Reservation*, IEEE Standard 802.1Qca-2015, 2016, pp. 1–120.

[16] *IEEE Standard for Local and Metropolitan Area Networks—Bridges and Bridged Networks—Amendment 26: Frame Preemption*, IEEE Standard 802.1Qbu-2016, 2016, pp. 1–52.
[17] *IEEE Standard for Local and Metropolitan Area Networks–Frame Replication and Elimination for Reliability*, IEEE Standard 802.1CB-2017, 2017, pp. 1–102.
[18] *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks–Amendment 29: Cyclic Queuing and Forwarding*, Standard IEEE 802.1Qch-2017, 2017, pp. 1–30.
[19] G. Patti, L. L. Bello, and L. Leonardi, "Deadline-aware online scheduling of TSN flows for automotive applications," *IEEE Trans. Ind. Informat.*, vol. 19, no. 4, pp. 5774–5784, Apr. 2023.
[20] M. Kim, J. Min, D. Hyeon, and J. Paek, "TAS scheduling for real-time forwarding of emergency event traffic in TSN," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2020, pp. 1111–1113.
[21] J. Xue, G. Shou, Y. Liu, Y. Hu, and Z. Guo, "Time-aware traffic scheduling with virtual queues in time-sensitive networking," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, May 2021, pp. 604–607.
[22] Y. Zhang, Q. Xu, S. Wang, Y. Chen, L. Xu, and C. Chen, "Scalable no-wait scheduling with flow-aware model conversion in time-sensitive networking," in *Proc. IEEE Global Commun. Conf.*, Dec. 2022, pp. 413–418.
[23] Z. Cao, Q. Liu, D. Liu, and Y. Hu, "Enhanced system design and scheduling strategy for switches in time-sensitive networking," *IEEE Access*, vol. 9, pp. 42621–42634, 2021.
[24] Y. Zhang, J. Wu, M. Liu, and A. Tan, "TSN-based routing and scheduling scheme for industrial Internet of Things in underground mining," *Eng. Appl. Artif. Intell.*, vol. 115, Oct. 2022, Art. no. 105314.
[25] Y. Zhang, Q. Xu, L. Xu, C. Chen, and X. Guan, "Efficient flow scheduling for industrial time-sensitive networking: A divisibility theory-based method," *IEEE Trans. Ind. Informat.*, vol. 18, no. 12, pp. 9312–9323, Dec. 2022.
[26] W. Han, Y. Li, and C. Yin, "A traffic scheduling algorithm combined with ingress shaping in TSN," in *Proc. 14th Int. Conf. Wireless Commun. Signal Process. (WCSP)*, Nov. 2022, pp. 586–591.
[27] M. Wei and S. Yang, "A network scheduling method for convergence of industrial wireless network and TSN," in *Proc. 17th Int. Conf. Ubiquitous Inf. Manage. Commun. (IMCOM)*, Jan. 2023, pp. 1–6.
[28] F. Dürr and N. G. Nayak, "No-wait packet scheduling for IEEE time-sensitive networks (TSN)," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, Oct. 2016, pp. 1–15.
[29] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks," in *Proc. 24th Int. Conf. Real-Time Netw. Syst.*, Oct. 2016, pp. 183–192.
[30] X. Jin, C. Xia, N. Guan, C. Xu, D. Li, Y. Yin, and P. Zeng, "Real-time scheduling of massive data in time sensitive networks with a limited number of schedule entries," *IEEE Access*, vol. 8, pp. 6751–6767, 2020.
[31] R. Serna Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1 Qbv gate control list synthesis using array theory encoding," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2018, pp. 13–24.
[32] A. Alnajim, S. Salehi, and C. Shen, "Incremental path-selection and scheduling for time-sensitive networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2019, pp. 1–6.
[33] A. A. Syed, S. Ayaz, T. Leinmüller, and M. Chandra, "Fault-tolerant dynamic scheduling and routing for TSN based in-vehicle networks," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Nov. 2021, pp. 72–75.
[34] Cplex, IBM ILOG, "V12.1: Users manual for CPLEX," *Int. Bus. Mach. Corp.*, vol. 46, no. 53, p. 157, 2009.
[35] Gurobi Optimization, LLC. (2021). *Gurobi Optimizer Reference Manual*. [Online]. Available: https://www.gurobi.com
[36] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Proc. Theory Pract. Softw., Int. Conf. Tools Algorithms Construct. Anal. Syst.*, 2008, pp. 337–340.
[37] L. Perron and V. Furnon. (Jul. 2019). *OR-Tools*. Google. [Online]. Available: https://developers.google.com/optimization/
[38] E. Alpaydin, *Introduction to Machine Learning*. Cambridge, MA, USA: MIT Press, 2020.
[39] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[40] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr, "Scaling TSN scheduling for factory automation networks," in *Proc. 16th IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, Apr. 2020, pp. 1–8.
[41] W. Steiner, S. S. Craciunas, and R. S. Oliver, "Traffic planning for time-sensitive communication," *IEEE Commun. Standards Mag.*, vol. 2, no. 2, pp. 42–47, Jun. 2018.
[42] H.-J. Kim, M.-H. Choi, M.-H. Kim, and S. Lee, "Development of an Ethernet-based heuristic time-sensitive networking scheduling algorithm for real-time in-vehicle data transmission," *Electronics*, vol. 10, no. 2, p. 157, Jan. 2021.
[43] H. J. Kim, K. C. Lee, and S. Lee, "A genetic algorithm based scheduling method for automotive Ethernet," in *Proc. 47th Annu. Conf. IEEE Ind. Electron. Soc.*, Oct. 2021, pp. 1–5.
[44] H.-J. Kim, K.-C. Lee, M.-H. Kim, and S. Lee, "Optimal scheduling of time-sensitive networks for automotive Ethernet based on genetic algorithm," *Electronics*, vol. 11, no. 6, p. 926, Mar. 2022.
[45] F. Ansah, M. A. Abid, and H. de Meer, "Schedulability analysis and GCL computation for time-sensitive networks," in *Proc. IEEE 17th Int. Conf. Ind. Informat. (INDIN)*, vol. 1, Jul. 2019, pp. 926–932.
[46] H. Wang, Z. Zhao, and J. Wei, "Adaptive scheduling algorithm based on time aware shaper," in *Proc. 5th Int. Conf. Adv. Electron. Mater., Comput. Softw. Eng. (AEMCSE)*, Apr. 2022, pp. 555–562.
[47] J. Min, M. Oh, W. Kim, H. Seo, and J. Paek, "Evaluation of metaheuristic algorithms for TAS scheduling in time-sensitive networking," in *Proc. 13th Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, Oct. 2022, pp. 809–812.
[48] M. Vlk, K. Brejchová, Z. Hanzálek, and S. Tang, "Large-scale periodic scheduling in time-sensitive networks," *Comput. Oper. Res.*, vol. 137, Jan. 2022, Art. no. 105512.
[49] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem-proving," *Commun. ACM*, vol. 5, no. 7, pp. 394–397, Jul. 1962.
[50] X. Wang, H. Yao, T. Mai, T. Nie, L. Zhu, and Y. Liu, "Deep reinforcement learning aided no-wait flow scheduling in time-sensitive networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2022, pp. 812–817.
[51] M. Vlk, Z. Hanzálek, K. Brejchová, S. Tang, S. Bhattacharjee, and S. Fu, "Enhancing schedulability and throughput of time-triggered traffic in IEEE 802.1 Qbv time-sensitive networks," *IEEE Trans. Commun.*, vol. 68, no. 11, pp. 7023–7038, Nov. 2020.
[52] N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Window-based schedule synthesis for industrial IEEE 802.1 Qbv TSN networks," in *Proc. 16th IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, Apr. 2020, pp. 1–4.
[53] P.-J. Chaine, M. Boyer, C. Pagetti, and F. Wartel, "Egress-TT configurations for TSN networks," in *Proc. 30th Int. Conf. Real-Time Netw. Syst.*, Jun. 2022, p. 5869.
[54] D. Bujosa, M. Ashjaei, A. V. Papadopoulos, T. Nolte, and J. Proenza, "HERMES: Heuristic multi-queue scheduler for TSN time-triggered traffic with zero reception jitter capabilities," in *Proc. 30th Int. Conf. Real-Time Netw. Syst.*, Jun. 2022, p. 7080.
[55] N. Reusch, S. S. Craciunas, and P. Pop, "Dependability-aware routing and scheduling for time-sensitive networking," *IET Cyber-Phys. Syst., Theory Appl.*, vol. 7, no. 3, pp. 124–146, Sep. 2022.
[56] N. G. Nayak, "Scheduling & routing time-triggered traffic in time-sensitive networks," Ph.D. dissertation, Graduate School Excellence Adv. Manuf. Eng. (GSaME), Univ. Stuttgart, Stuttgart, Germany, Tech. Rep., 2018.
[57] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design optimisation of cyber-physical distributed systems using IEEE time-sensitive networks," *IET Cyber-Phys. Syst., Theory Appl.*, vol. 1, no. 1, pp. 86–94, Dec. 2016.
[58] E. Li, F. He, L. Zhao, and X. Zhou, "A SDN-based traffic bandwidth allocation method for time sensitive networking in avionics," in *Proc. IEEE/AIAA 38th Digit. Avionics Syst. Conf. (DASC)*, Sep. 2019, pp. 1–7.
[59] A. C. T. dos Santos, B. Schneider, and V. Nigam, "TSNSCHED: Automated schedule generation for time sensitive networking," in *Proc. Formal Methods Comput. Aided Design*, 2019, pp. 69–77.
[60] A. Varga, *OMNeT++*. Berlin, Germany: Springer, 2010, pp. 35–59.
[61] A. C. T. D. Santos, "TSNsched: Automated schedule generation for time sensitive networking," Ph.D. dissertation, Centro de Informática, Universidade Federal da Paraíba, Paraíba, Brazil, 2020.
[62] B. Houtan, M. Ashjaei, M. Daneshtalab, M. Sjödin, and S. Mubeen, "Synthesising schedules to improve QoS of best-effort traffic in TSN networks," in *Proc. 29th Int. Conf. Real-Time Netw. Syst.*, Apr. 2021, pp. 68–77.

[63] W. Zhou and Z. Li, "Implementation and evaluation of SMT-based real-time communication scheduling for IEEE 802.1 Qbv in next-generation in-vehicle network," in *Proc. 2nd Int. Conf. Inf. Technol. Comput. Appl. (ITCA)*, Dec. 2020, pp. 457–461.

[64] M. Barzegaran, N. Reusch, L. Zhao, S. S. Craciunas, and P. Pop, "Real-time traffic guarantees in heterogeneous time-sensitive networks," in *Proc. 30th Int. Conf. Real-Time Netw. Syst.*, Jun. 2022, p. 4657.

[65] J. Pei, Y. Hu, L. Tian, M. Li, and Z. Li, "A hybrid traffic scheduling strategy for time-sensitive networking," *Electronics*, vol. 11, no. 22, p. 3762, Nov. 2022.

[66] X. Yao, Z. Gan, Y. Chen, L. Guo, and W. Wang, "Hybrid flow scheduling with additional simple compensation mechanisms in time-sensitive networks," in *Proc. IEEE 6th Adv. Inf. Technol., Electron. Autom. Control Conf. (IAEAC)*, Oct. 2022, pp. 1315–1320.

[67] S. Wang, Q. Xu, Y. Zhang, L. Xu, and C. Chen, "Hybrid traffic scheduling based on adaptive time slot slicing in time-sensitive networking," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Aug. 2022, pp. 1–7.

[68] Z. Huang, H. Zhu, H. Zhang, and T. Huang, "A scalable heuristic time-sensitive traffic scheduling algorithm for in-vehicle network," in *Proc. 5th Int. Conf. Hot Inf.-Centric Netw. (HotICN)*, Nov. 2022, pp. 111–118.

[69] S. S. Craciunas and R. S. Oliver, "Out-of-sync schedule robustness for time-sensitive networks," in *Proc. 17th IEEE Int. Conf. Factory Commun. Syst. (WFCS)*, Jun. 2021, pp. 75–82.

[70] Z. Feng, M. Cai, and Q. Deng, "An efficient pro-active fault-tolerance scheduling of IEEE 802.1 Qbv time-sensitive network," *IEEE Internet Things J.*, vol. 9, no. 16, pp. 14501–14510, Aug. 2022.

[71] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Routing and scheduling of time-triggered traffic in time-sensitive networks," *IEEE Trans. Ind. Informat.*, vol. 16, no. 7, pp. 4525–4534, Jul. 2020.

[72] N. Reusch, P. Pop, and S. S. Craciunas, "Work-in-progress: Safe and secure configuration synthesis for TSN using constraint programming," in *Proc. IEEE Real-Time Syst. Symp. (RTSS)*, Dec. 2020, pp. 387–390.

[73] V. Gavrilut, B. Zarrin, P. Pop, and S. Samii, "Fault-tolerant topology and routing synthesis for IEEE time-sensitive networking," in *Proc. 25th Int. Conf. Real-Time Netw. Syst.*, Oct. 2017, pp. 267–276.

[74] Z. Feng, Q. Deng, M. Cai, and J. Li, "Efficient reservation-based fault-tolerant scheduling for IEEE 802.1 Qbv time-sensitive networking," *J. Syst. Archit.*, vol. 123, Feb. 2022, Art. no. 102381.

[75] R. Dobrin, N. Desai, and S. Punnekkat, "On fault-tolerant scheduling of time sensitive networks," in *Proc. Int. Workshop Secur. Dependability Crit. Embedded Real-Time Syst.*, vol. 73, 2019, pp. 5:1–5:13.

[76] M. L. Raagaard, P. Pop, M. Gutiérrez, and W. Steiner, "Runtime reconfiguration of time-sensitive networking (TSN) schedules for fog computing," in *Proc. IEEE Fog World Congr. (FWC)*, Oct. 2017, pp. 1–6.

[77] Z. Pang, X. Huang, Z. Li, S. Zhang, Y. Xu, H. Wan, and X. Zhao, "Flow scheduling for conflict-free network updates in time-sensitive software-defined networks," *IEEE Trans. Ind. Informat.*, vol. 17, no. 3, pp. 1668–1678, Mar. 2021.

[78] Y. Wang, F. Wang, W. Wang, X. Tan, J. Wen, Y. Wang, and P. Lin, "Design and implementation of traffic scheduling algorithm for time-sensitive network," in *Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcast. (BMSB)*, Jun. 2022, pp. 1–6.

[79] C. Gärtner, A. Rizk, B. Koldehofe, R. Guillaume, R. Kundel, and R. Steinmetz, "On the incremental reconfiguration of time-sensitive networks at runtime," in *Proc. IFIP Netw. Conf.*, Jun. 2022, pp. 1–9.

[80] C. Gärtner, A. Rizk, B. Koldehofe, R. Guillaume, R. Kundel, and R. Steinmetz, "Fast incremental reconfiguration of dynamic time-sensitive networks at runtime," *Comput. Netw.*, vol. 224, Apr. 2023, Art. no. 109606.

[81] Q. Li, D. Li, X. Jin, Q. Wang, and P. Zeng, "A simple and efficient time-sensitive networking traffic scheduling method for industrial scenarios," *Electronics*, vol. 9, no. 12, p. 2131, Dec. 2020.

[82] J. Dai, Z. Wang, and L. Zhong, "Research on gating scheduling of time sensitive network based on constraint strategy," *J. Phys., Conf. Ser.*, vol. 1920, no. 1, May 2021, Art. no. 012089.

[83] T. Feng and H. Yang, "SMT-based task- and network-level static schedule for time sensitive network," in *Proc. Int. Conf. Commun., Inf. Syst. Comput. Eng. (CISCE)*, May 2021, pp. 764–770.

[84] M. Barzegaran, B. Zarrin, and P. Pop, "Quality-of-control-aware scheduling of communication in TSN-based fog computing platforms using constraint programming," in *Proc. Workshop Fog Comput. IoT*, 2020, pp. 3:1–3:9.

[85] M. Barzegaran and P. Pop, "Communication scheduling for control performance in TSN-based fog computing platforms," *IEEE Access*, vol. 9, pp. 50782–50797, 2021.

[86] M. Barzegaran, "Configuration optimization of fog computing platforms for control applications," Ph.D. dissertation, Dept. Appl. Math. Comput. Sci., Tech. Univ. Denmark, Lyngby, Denmark, 2021.

[87] S. D. McLean, E. A. Juul Hansen, P. Pop, and S. S. Craciunas, "Configuring ADAS platforms for automotive applications using metaheuristics," *Frontiers Robot. AI*, vol. 8, pp. 1–15, Jan. 2022.

[88] A. Arestova, W. Baron, K. J. Hielscher, and R. German, "ITANS: Incremental task and network scheduling for time-sensitive networks," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 369–387, 2022.

[89] X. Jin, C. Xia, N. Guan, and P. Zeng, "Joint algorithm of message fragmentation and no-wait scheduling for time-sensitive networks," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 2, pp. 478–490, Feb. 2021.

[90] T. Park, S. Samii, and K. G. Shin, "Design optimization of frame preemption in real-time switched Ethernet," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 420–425.

[91] D. Ginthör, R. Guillaume, J. von Hoyningen-Huene, M. Schüngel, and H. D. Schotten, "End-to-end optimized joint scheduling of converged wireless and wired time-sensitive networks," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2020, pp. 222–229.

[92] J. Lin, W. Li, X. Feng, S. Zhan, J. Feng, J. Cheng, T. Wang, Q. Li, Y. Wang, F. Li, and B. Tang, "Rethinking the use of network cycle in time-sensitive networking (TSN) flow scheduling," in *Proc. IEEE/ACM 30th Int. Symp. Quality Service (IWQoS)*, Jun. 2022, pp. 1–11.

[93] M. H. Farzaneh, S. Kugele, and A. Knoll, "A graphical modeling tool supporting automated schedule synthesis for time-sensitive networking," in *Proc. 22nd IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2017, pp. 1–8.

[94] J. Tu, Q. Xu, L. Xu, and C. Chen, "SSL-SP: A semi-supervised-learning-based stream partitioning method for scale iterated scheduling in time-sensitive networks," in *Proc. 22nd IEEE Int. Conf. Ind. Technol. (ICIT)*, vol. 1, Mar. 2021, pp. 1182–1187.

[95] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Stability-aware integrated routing and scheduling for control applications in Ethernet networks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 682–687.

[96] J. Falk, F. Dürr, and K. Rothermel, "Exploring practical limitations of joint routing and scheduling for TSN with ILP," in *Proc. IEEE 24th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2018, pp. 136–146.

[97] H. Nie, S. Li, and Y. Liu, "An enhanced routing and scheduling mechanism for time-triggered traffic with large period differences in time-sensitive networking," *Appl. Sci.*, vol. 12, no. 9, p. 4448, Apr. 2022.

[98] Y. Huang, S. Wang, T. Huang, B. Wu, Y. Wu, and Y. Liu, "Online routing and scheduling for time-sensitive networks," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2021, pp. 272–281.

[99] L. Xu, Q. Xu, J. Tu, J. Zhang, Y. Zhang, C. Chen, and X. Guan, "Learning-based scalable scheduling and routing co-design with stream similarity partitioning for time-sensitive networking," *IEEE Internet Things J.*, vol. 9, no. 15, pp. 13353–13363, Aug. 2022.

[100] M. Pahlevan, N. Tabassam, and R. Obermaisser, "Heuristic list scheduler for time triggered traffic in time sensitive networks," *ACM SIGBED Rev.*, vol. 16, no. 1, pp. 15–20, Feb. 2019.

[101] L. Xu, Q. Xu, Y. Zhang, J. Zhang, and C. Chen, "Co-design approach of scheduling and routing in time sensitive networking," in *Proc. IEEE Conf. Ind. Cyberphys. Syst. (ICPS)*, vol. 1, Jun. 2020, pp. 111–116.

[102] F. Smirnov, M. Glaß, F. Reimann, and J. Teich, "Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.

[103] A. Arestova, K.-S. J. Hielscher, and R. German, "Design of a hybrid genetic algorithm for time-sensitive networking," in *Proc. Int. Conf. Meas., Modeling Eval. Comput. Syst.*, 2020, pp. 99–117.

[104] M. Pahlevan and R. Obermaisser, "Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks," in *Proc. IEEE 23rd Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2018, pp. 337–344.

[105] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, Jan. 1983.

[106] A. M. Kentis, M. S. Berger, and J. Soler, "Effects of port congestion in the gate control list scheduling of time sensitive networks," in *Proc. 8th Int. Conf. Netw. Future (NOF)*, Nov. 2017, pp. 138–140.

[107] Y. Wang, J. Chen, W. Ning, H. Yu, S. Lin, Z. Wang, G. Pang, and C. Chen, "A time-sensitive network scheduling algorithm based on improved ant colony optimization," *Alexandria Eng. J.*, vol. 60, no. 1, pp. 107–114, Feb. 2021.

[108] J. Falk, F. Dürr, and K. Rothermel, "Time-triggered traffic planning for data networks with conflict graphs," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, Apr. 2020, pp. 124–136.

[109] M. Vlk, Z. Hanzálek, and S. Tang, "Constraint programming approaches to joint routing and scheduling in time-sensitive networks," *Comput. Ind. Eng.*, vol. 157, Jul. 2021, Art. no. 107317.

[110] B. Caddell, "Joint routing and scheduling with SMT," B.Sc. thesis, Inst. Parallel Distrib. Syst., Univ. Stuttgart, Stuttgart, Germany, 2018.

[111] X. He, X. Zhuge, F. Dang, W. Xu, and Z. Yang, "DeepScheduler: Enabling flow-aware scheduling in time-sensitive networking," in *Proc. IEEE INFOCOM*, Mar. 2023, pp. 1–4.

[112] F. Pozo, G. Rodriguez-Navas, and H. Hansson, "Schedule reparability: Enhancing time-triggered network recovery upon link failures," in *Proc. IEEE 24th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2018, pp. 147–156.

[113] A. A. Atallah, G. B. Hamad, and O. A. Mohamed, "Fault-resilient topology planning and traffic configuration for IEEE 802.1 Qbv TSN networks," in *Proc. IEEE 24th Int. Symp. On-Line Test. Robust Syst. Design (IOLTS)*, Jul. 2018, pp. 151–156.

[114] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Reliability-aware scheduling and routing for messages in time-sensitive networking," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5, pp. 1–24, Sep. 2021.

[115] A. A. Syed, S. Ayaz, T. Leinmüller, and M. Chandra, "Fault-tolerant static scheduling and routing for in-vehicle networks," in *Proc. 32nd Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2022, pp. 273–279.

[116] A. A. Syed, S. Ayaz, T. Leinmüller, and M. Chandra, "MIP-based joint scheduling and routing with load balancing for TSN based in-vehicle networks," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Dec. 2020, pp. 1–7.

[117] A. A. Syed, S. Ayaz, T. Leinmüller, and M. Chandra, "Network coding based fault-tolerant dynamic scheduling and routing for in-vehicle networks," *J. Netw. Syst. Manage.*, vol. 31, no. 1, p. 27, Jan. 2023.

[118] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "ASIL-decomposition based routing and scheduling in safety-critical time-sensitive networking," in *Proc. IEEE 27th Real-Time Embedded Technol. Appl. Symp. (RTAS)*, May 2021, pp. 184–195.

[119] *Road Vehicles: Functional Safety*, Standard ISO 26262, 2018.

[120] Y. Zhou, "Synthesis of safety-critical real-time systems," Ph.D. dissertation, Dept. Comput. Inf. Sci., Linköping Univ., Linköping, Sweden, 2022.

[121] Y. Zhou, S. Samii, P. Eles, and Z. Peng, "Time-triggered scheduling for time-sensitive networking with preemption," in *Proc. 27th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2022, pp. 262–267.

[122] H. Li, H. Cheng, and L. Yang, "Reliable routing and scheduling in time-sensitive networks," in *Proc. 17th Int. Conf. Mobility, Sens. Netw. (MSN)*, Dec. 2021, pp. 806–811.

[123] V. Gavrilut, L. Zhao, M. L. Raagaard, and P. Pop, "AVB-aware routing and scheduling of time-triggered traffic for TSN," *IEEE Access*, vol. 6, pp. 75229–75243, 2018.

[124] S. M. Laursen, P. Pop, and W. Steiner, "Routing optimization of AVB streams in TSN networks," *SIGBED Rev.*, vol. 13, no. 4, p. 4348, 2016.

[125] V. Gavrilut and P. Pop, "Scheduling in time sensitive networks (TSN) for mixed-criticality industrial applications," in *Proc. 14th IEEE Int. Workshop Factory Commun. Syst. (WFCS)*, Jun. 2018, pp. 1–4.

[126] M. L. Raagaard and P. Pop, "Optimization algorithms for the scheduling of IEEE 802.1 time-sensitive networking (TSN)," Tech. Univ. Denmark, Lyngby, Denmark, Tech. Rep. 1, 2017.

[127] C. Chuang, T. Yu, C. Lin, A. Pang, and T. Hsieh, "Online stream-aware routing for TSN-based industrial control systems," in *Proc. 25th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, vol. 1, Sep. 2020, pp. 254–261.

[128] V. Gavriluţ and P. Pop, "Traffic-type assignment for TSN-based mixed-criticality cyber-physical systems," *ACM Trans. Cyber-Phys. Syst.*, vol. 4, no. 2, pp. 1–27, Apr. 2020.

[129] A. Berisa, L. Zhao, S. S. Craciunas, M. Ashjaei, S. Mubeen, M. Daneshtalab, and M. Sjödin, "AVB-aware routing and scheduling for critical traffic in time-sensitive networks with preemption," in *Proc. 30th Int. Conf. Real-Time Netw. Syst.*, Jun. 2022, pp. 207–218.

[130] Y. Li, J. Jiang, and S. H. Hong, "Joint traffic routing and scheduling algorithm eliminating the nondeterministic interruption for TSN networks used in IIoT," *IEEE Internet Things J.*, vol. 9, no. 19, pp. 18663–18680, Oct. 2022.

[131] L. Yang, Y. Wei, F. R. Yu, and Z. Han, "Joint routing and scheduling optimization in time-sensitive networks using graph-convolutional-network-based deep reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 23, pp. 23981–23994, Dec. 2022.

[132] E. Schweissguth, D. Timmermann, H. Parzyjegla, P. Danielis, and G. Mühl, "ILP-based routing and scheduling of multicast realtime traffic in time-sensitive networks," in *Proc. IEEE 26th Int. Conf. Embedded Real-Time Comput. Syst. Appl. (RTCSA)*, Aug. 2020, pp. 1–11.

[133] C. Li, C. Zhang, W. Zheng, X. Wen, Z. Lu, and J. Zhao, "Joint routing and scheduling for dynamic applications in multicast time-sensitive networks," in *Proc. IEEE Int. Conf. Commun. Workshops*, Jun. 2021, pp. 1–6.

[134] Q. Yu and M. Gu, "Adaptive group routing and scheduling in multicast time-sensitive networks," *IEEE Access*, vol. 8, pp. 37855–37865, 2020.

[135] A. A. Syed, S. Ayaz, T. Leinmüller, and M. Chandra, "Dynamic scheduling and routing for TSN based in-vehicle networks," in *Proc. IEEE Int. Conf. Commun. Workshops*, Jun. 2021, pp. 1–6.

[136] Q. Yu, H. Wan, X. Zhao, Y. Gao, and M. Gu, "Online scheduling for dynamic VM migration in multicast time-sensitive networks," *IEEE Trans. Ind. Informat.*, vol. 16, no. 6, pp. 3778–3788, Jun. 2020.

[137] L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for Steiner trees," *Acta Inf.*, vol. 15, no. 2, pp. 141–145, 1981.

[138] J. Li, H. Xiong, Q. Li, F. Xiong, and J. Feng, "Run-time reconfiguration strategy and implementation of time-triggered networks," *Electronics*, vol. 11, no. 9, p. 1477, May 2022.

[139] N. G. Nayak, F. Dürr, and K. Rothermel, "Incremental flow scheduling and routing in time-sensitive software-defined networks," *IEEE Trans. Ind. Informat.*, vol. 14, no. 5, pp. 2066–2075, May 2018.

[140] R. Mahfouzi, A. Aminifar, S. Samii, A. Rezine, P. Eles, and Z. Peng, "Breaking silos to guarantee control stability with communication over Ethernet TSN," *IEEE Des. Test. IEEE Des. Test. Comput.*, vol. 38, no. 5, pp. 48–56, Oct. 2021.

[141] Y. Zheng, S. Wang, S. Yin, B. Wu, and Y. Liu, "Mix-flow scheduling for concurrent multipath transmission in time-sensitive networking," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC Workshops)*, Jun. 2021, pp. 1–6.

[142] D. Yang, K. Gong, J. Ren, W. Zhang, W. Wu, and H. Zhang, "TC-flow: Chain flow scheduling for advanced industrial applications in time-sensitive networks," *IEEE Netw.*, vol. 36, no. 2, pp. 16–24, Mar. 2022.

[143] K. Gong, D. Yang, W. Zhang, and J. Ren, "An efficient scheduling approach for multi-level industrial chain flows in time-sensitive networking," *Comput. Netw.*, vol. 221, Feb. 2023, Art. no. 109516.

[144] D. Hellmanns, L. Haug, M. Hildebrand, F. Dürr, S. Kehrer, and R. Hummen, "How to optimize joint routing and scheduling models for TSN using integer linear programming," in *Proc. 29th Int. Conf. Real-Time Netw. Syst.*, Apr. 2021, pp. 100–111.

[145] S. Bhattacharjee, K. Alexandris, E. Hansen, P. Pop, and T. Bauschert, "Latency-aware function placement, routing, and scheduling in TSN-based industrial networks," in *Proc. IEEE Int. Conf. Commun.*, May 2022, pp. 4248–4254.

[146] A. Perrig, R. Canetti, D. Song, and J. D. Tygar, "Efficient and secure source authentication for multicast," in *Proc. Netw. Distrib. Syst. Secur. Symp. (NDSS)*, vol. 1, 2001, pp. 35–46.

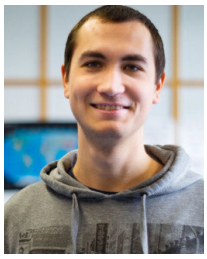
[147] *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing*, IEEE Standard 802.1Qci-2017, 2017, pp. 1–65.

[148] *Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges*, IEEE Standard 802.1D-1990, 1991, pp. 1–176.

[149] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*. London, U.K.: Pearson, 2016.

[150] *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks—Amendment 34: Asynchronous Traffic Shaping*, IEEE Standard 802.1Qcr-2020, 2020, pp. 1–151.

THOMAS STÜBER received the master's degree, in 2018. He is currently pursuing the Ph.D. degree with the Chair of Communication Networks of Prof. Dr. Habil. Michael Menth, University of Tübingen, Germany. He became part of the Communication Networks Research Group. His research interests include time-sensitive networking (TSN), scheduling, performance evaluation, and operations research.



LUKAS OSSWALD received the master's degree, in 2020. He is currently pursuing the Ph.D. degree with the Chair of Communication Networks of Prof. Dr. Habil. Michael Menth, University of Tübingen, Germany. He became part of the Communication Networks Research Group. His research interests include time-sensitive networking (TSN), admission control, and network configuration.

STEFFEN LINDNER received the master's degree, in 2019. He is currently pursuing the Ph.D. degree with the Chair of Communication Networks of Prof. Dr. Habil. Michael Menth, University of Tübingen, Germany. He became part of the Communication Networks Research Group. His research interests include software-defined networking, P4, and congestion management.

MICHAEL MENTH (Senior Member, IEEE) received the Diploma degree from The University of Texas at Austin, Austin, TX, USA, in 1998, the Ph.D. degree from Ulm University, Germany, in 2004, and the Habilitation degree from the University of Würzburg, Germany, in 2010. He has been the Chair Holder of Communication Networks, since 2010. He is currently a Professor with the Department of Computer Science, University of Tübingen, Germany. His special interests are performance analysis and optimization of communication networks, resilience and routing issues, as well as resource and congestion management. His recent research interests include network softwarization, in particular P4-based data plane programming, time-sensitive networking (TSN), the Internet of Things, and internet protocols. He contributes to standardization bodies, mainly to the IETF.

1.3 Performance Comparison of Offline Scheduling Algorithms for the Time-Aware Shaper (TAS)

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

Performance Comparison of Offline Scheduling Algorithms for the Time-Aware Shaper (TAS)

***Abstract*— Time-Sensitive Networking (TSN) is an emerging technology that enables deterministic and reliable transmission in bridged Ethernet networks. The enhancement for scheduled traffic defined in IEEE 802.1Qbv [1] allows to implement time-aware shaping (TAS) which grants periodic slices for transmission to various priority queues of a bridge. TAS is an enabler for traffic scheduling, i.e., frame transmissions of periodic streams at senders and the TAS on intermediate bridges are configured such that these frames experience no loss and hardly any queuing delay. Thereby, deterministic bounds on delay and jitter can be guaranteed to such streams. However, the standard does not provide an algorithm to compute transmission schedules. Therefore, more than 100 research works [2] propose various algorithms for computing such schedules. Nevertheless, there are still many challenges to solve in this area. In this work, we implement 11 of these algorithms and compare their performance under various conditions with regard to schedule quality and runtime. It reveals that the performance of the algorithms varies a lot and points out their shortcomings. The set of problem instances for this study covers a wide range of parameters and is released to the public so that the performance of new algorithms can be easily compared to those in this study.**

I. Introduction

Time-Sensitive Networking (TSN) is a set of IEEE standards that enhance Ethernet bridging for deterministic transmission. The enhancement for scheduled traffic defined in IEEE 802.1Qbv [1] allows the time-controlled shaping of up to eight priority queues in bridges, which is commonly denoted as time-aware shaping (TAS). For this purpose, a periodic gate control list (GCL) defines which queues are eligible for transmission in specific time intervals. TAS is intended to protect time-triggered (TT) streams against queuing delay induced by other traffic. TT streams are periodic and have real-time requirements such as bounded latency or jitter. For traffic scheduling, the periodic transmission times of TT streams and the GCLs on intermediate bridges are configured such that their frames experience no loss and hardly any queuing delay. This configuration is based on a schedule whose computation is not standardized by the IEEE. However, there is a wide range of algorithms to calculate such schedules (scheduling algorithms), but none of them is sufficient for application in practice. Thus, there is still a lot of work to be done in this area. Moreover, it is hard to compare these algorithms as most research works evaluate only their own algorithm based on an own set of problem instances. In addition, the problem instances are not published so that results for new algorithms cannot be compared without implementing existing algorithms, which requires additional effort.

A. Contribution

We explain the threefold contribution of this work and its motivation. First, we compare the performance of 11 well-known offline scheduling algorithms with regard to schedule quality and runtime. We implemented them and conducted insightful parameter studies. Thus, this work informs engineers and other practitioners about benefits and drawbacks of existing scheduling algorithms. Second, we published the investigated problem instances on GitHub[1]. If new algorithms are developed, they may be tested on the same problem instances so that their performance can be easily compared to the state of the art. Thereby, researchers can compare the performance of new methods to existing works at an early stage without the need to reimplement a set of algorithms from the literature. Third, this is the first reproducible evaluation about scheduling algorithms for the TAS in the literature. The published problem instances can be used to validate the reported results by independent researchers. Thus, this work marks an important step towards better scientific practices in the domain of TAS scheduling.

The performance evaluation considers solving times of problem instances for first and final solutions. Runtime scalability is tested with computation and memory limits on problem instances with a different number of streams and bridges. The performance gain through multi-threading is studied. The ability of algorithms to find existing solutions is investigated, which is relevant for heuristic approaches. Finally, schedule quality is measured in terms of average end-to-end delay of frames including queuing, relative to average end-to-end delay on a fastest path without any queuing.

B. Comparison with Similar Works

Nasrallah et al. [3] conduct an extensive performance comparison of TAS and Asynchronous Traffic Shaping. They configure the TAS by reserving a fixed proportion of the cycle time for scheduled traffic. Therefore, they do not use or compare offline scheduling algorithms for the TAS. Additionally, their traffic model lacks deadlines and streams with different periods, and the problem instances are not published.

Only recently, the authors of [4] and [5] compared their three algorithms with the one in [6] and defined a well described benchmarking methodology [7]. Our work goes beyond this approach as we consider 11 algorithms based on different optimization methods from different authors and

[1] [redacted]

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

the developed benchmarking methodology is more comprehensive. For example, streams with different periods are investigated, larger problem instances are studied, queuing and GCL lengths are considered, to name a few. Additionally, the periods considered in [7] are significantly smaller than 1 ms. Thus, the presented problem instances do not represent industrial [8], automotive [9], or aerospace [10] use cases in contrast to the problem instances proposed in this work. We also study scheduling under challenging conditions and focus on a systematic construction of experiment series.

The remainder of this work is structured as follows. Section II gives some background on TSN technology, it introduces the scheduling problem of TAS in TSN, and reviews common solving methods. Then, the 11 scheduling algorithms considered in this work are summarized and compared in Section III. The methodology for the performance evaluation and the set of problem instances are described in Section V. Section VI presents the evaluation results. We summarize the findings and discuss future research directions in Section VII. Section VIII concludes the paper.

II. Preliminaries

We first summarize TSN technology with focus on time-aware shaping (TAS). Then, we define the scheduling problem of TAS in TSN. Finally, we review common solving methods for the scheduling problem.

A. Time-Sensitive Networking (TSN)

Time-Sensitive Networking (TSN) is a set of standards for reliable and deterministic data transmission over Ethernet networks. These standards cover time synchronization, traffic shaping and scheduling, as well as network management. For traffic scheduling in TSN, bridges and end stations require a common understanding of time such that their actions can be executed in a coordinated fashion. Therefore, every device is equipped with a clock. These clocks are synchronized with the generalized Precision Time Protocol (gPTP) defined in IEEE 802.1AS [11]. It enables sub-microsecond precision for networks with a diameter of at most seven hops. The enhancement for scheduled traffic introduced in IEEE 802.1Qbv [1] proposes mechanisms for time-aware shaping (TAS). We explain TAS using Figure 1. Every egress port is equipped with up to eight egress queues that correspond to eight priorities. Frames are assigned to these queues according to their priority which is indicated by the Priority Code Point (PCP) field in the VLAN tag of the Ethernet header. Frames within a queue are served in a first-in-first-out (FIFO) manner so that frames of a stream are not reordered. Every egress queue is guarded by a so-called transmission gate. The gates open and close periodically, which is controlled by a so-called gate control list (GCL) per egress port. A GCL entry is a pair of a time interval and a bit vector indicating whether the gates are opened or closed during that time interval. The GCL is executed periodically. Only queues with an open gate are allowed to send frames. Transmission selection is assumed to be strict priority, i.e., frames are dispatched from the highest priority queue with an open gate and a frame waiting for transmission. We point out that the number of available entries per GCL is limited in hardware bridges.

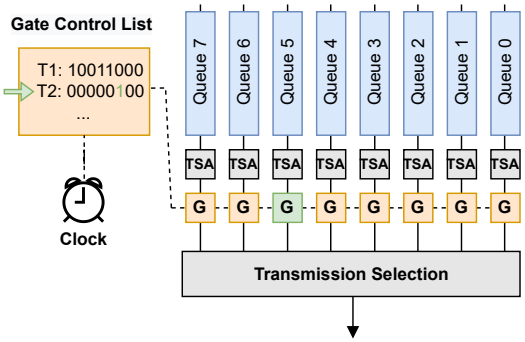


Fig. 1. Model of an egress port in TSN implementing the enhancement for scheduled traffic.

B. The Scheduling Problem of TAS in TSN

In TSN, senders and receivers of a frame are denoted as talkers and listeners. Traffic scheduling requires that periodic transmission times of TT streams at talkers and GCL entries on intermediate bridges are configured such that delay bounds of these TT streams are met when their frames are received by listeners. This configuration requires a schedule comprising the periodic frame transmission times and GCLs. The periodic nature of TT streams and GCLs implies that the schedule is executed periodically.

In the following, we state a general version of the problem formally. A network topology can be modelled by a directed graph $G = (V, E)$. V is the set of vertices, i.e., the bridges and end stations. Each network device $d \in V$ is a 2-tuple (n_{GCL}, d_{proc}), where n_{GCL} is the number of available GCL entries per egress port, and d_{proc} is the processing delay of d. E is the set of edges, i.e., the directed links of the network. Egress ports of network devices and directed links have a one-to-one correspondence. Therefore we will identify egress ports with the corresponding attached links. Every link $l \in E$ is itself a 4-tuple $(src, dest, d_{prop}, b)$, where src and $dest$ are source and destination network devices of l, d_{prop} is the propagation delay of l, and b is the transmission bandwidth of the egress port l is attached to. A TT stream s is a 6-tuple $(v_{talker}, V_{listeners}, n_f, f_s, T_s, d)$, where v_{talker} is the talker of s, $V_{listeners}$ is the set of listeners of s, n_f is the maximum number of frames per period, f_s is the maximum frame size, T_s is the duration of a period, and d is the deadline of s. We remark that multiple periods of the same stream may be contained in a schedule and that d is relative to the start of a streams period, which may differ from the start of the schedule. A schedule is formally a mapping of frames to transmission offsets at their respective talkers, together with a mapping of egress ports to GCLs and a mapping of TT streams to traffic classes. We remark that many works assume the mapping to traffic classes as fixed before scheduling and do not leverage this degree of freedom.

The scheduling problem for the TAS can be stated as

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

follows: given a network topology G, a set of TT streams S, and possibly a routing map $R : S \to Path(G)$, compute a schedule such that all frames arrive before their respective deadlines at all of their respective listeners. In general, not all TT streams have the same period. Let H be the least common multiple of the periods of all TT streams. H is denoted as the *hyperperiod* in the literature and is the period of a schedule. Thus, all frames of a stream s are contained $\frac{H}{T_s}$ times in a schedule such that their transmission offsets are exactly T_s apart. Additionally, joint routing algorithms do not have a routing as input, but return such a mapping from streams to paths.

The challenge of computing such a schedule constitutes the scheduling problem of TAS in TSN. For instance, most research works assume that the traffic classes of the streams are given while others jointly compute traffic classes and schedules. Computing schedules in TSN is hard and the scheduling problem is known to be NP-complete [4]. That means that there is probably no algorithm with subexponential runtime that decides whether a valid schedule exists for a given problem instance. We remark that we only investigate the offline scheduling problem, i.e., all streams are known in advance and only a single schedule is computed. The online scheduling problem, i.e., streams are added and removed on the fly and schedules are updated accordingly, represents another problem with different performance metrics and algorithms and is beyond the scope of this work.

C. Solving Methods for the Scheduling Problem

Scheduling algorithms can be classified into exact and heuristic approaches. Exact approaches are guaranteed to find a schedule if one exists. If an optimization objective is given, they are able to find a schedule which minimizes or maximizes this objective among the set of all valid schedules. Additionally, such approaches can decide whether a problem instance is infeasible, i.e., whether no valid schedule exists. An Integer Linear Program (ILP) models a problem instance with linear inequalities of integer variables. Satisfiability modulo theories (SMT) models describe a problem instance with propositional logic and predicates from various theories, e.g., the theory of integer arithmetic. For both ILP and SMT models holds that each fulfilling variable assignment corresponds to a valid schedule for a problem instance, and vice versa. Additionally, ILP solvers are able to infer upper bounds for the gap between the objective value of the current solution and the unknown optimal solution during solving. Solving ILPs and SMTs is hard and may take a significant amount of time. Therefore, specialized solvers are used for this purpose. A general drawback of ILP approaches is that most ILP solvers require an expensive licence for commercial use while most SMT solvers and heuristics are free.

In contrast, heuristic approaches are designed for fast construction of reasonably good solutions. However, heuristics are not guaranteed to find a solution even though a valid schedule exists. Optimization heuristics may return suboptimal solutions and cannot decide whether a found solution is optimal. A summary of all kinds of scheduling heuristics considered in this work is out of scope. However, an important subroutine of most heuristics for TSN scheduling is "as soon as possible time tabling". Frames are scheduled one after another in some given order. Every frame is scheduled along the path from its talker to its listener. A resource conflict occurs when two frames are scheduled for the same resource at the same time, e.g., when two frames are scheduled to be simultaneously transmitted over the same link. At every hop on its path, a frame is scheduled at the earliest possible time such that resource conflicts with already scheduled frames are avoided. Additional constraints may be imposed by algorithms that allow queuing of frames in the egress queue.

III. Related Work

We give an overview of the 11 scheduling algorithms compared in this work. Table 1 compiles them including abbreviations used for them in this work and their features that are discussed in the following. We differentiate between exact and heuristic algorithms as discussed in Section II-C. Additionally, we classify algorithms with respect to their handling of stream paths. Approaches that consider stream paths as fixed prior to scheduling are denoted as scheduling w/o routing. Others calculating stream paths are denoted as joint routing and scheduling approaches. An incremental algorithm schedules streams one after another. The schedules of streams computed in earlier iterations are fixed and cannot be changed afterwards.

Algorithm	Method	Incremental	Routing	Queuing	Impl.
M2F [12]	Heur.	✓	-	✓	hard
Tabu [4]	Heur.	✓	-	-	easy
ConfGraph [13]	Heur.	✓	✓	-	hard
GenAlg [14]	Heur.	✓	✓	-	medium
GRASP [15]	Heur.	✓	-	✓	hard
HLS [16]	Heur.	✓	✓	-	easy
SMT-INC [17]	SMT	✓	-	✓	easy
SMT-DEC [17]	SMT	✓	-	✓	easy
ILP-NoWait [4]	ILP	-	-	-	easy
ILP-JR-1 [18]	ILP	-	✓	-	easy
ILP-JR-2 [5]	ILP	-	✓	✓	medium

TABLE 1
ALGORITHMS COMPARED IN THIS WORK AND THEIR FEATURES. THE LAST COLUMN INDICATES OUR SUBJECTIVE IMPRESSION ON IMPLEMENTATION COMPLEXITY.

A. Scheduling w/o Routing

Dürr et al. [4] consider the so-called no-wait scheduling problem, i.e., all frames are forwarded with zero-queuing. That means that the transmission times of frames at intermediate nodes are fully determined by the transmission times at the respective talkers. The flowspan is the time within all frames have arrived at their respective listeners relative to the start of a schedule repetition. They propose an ILP (ILP-NoWait) and a tabu search heuristic (Tabu) to compute transmission schedules that minimize the flowspan. Tabu resolves resource conflicts during time tabling by delaying the transmission time of a frame at its talker. It optimizes the frame order for the time tabling algorithm to reduce resource conflicts.

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

Craciunas et al. [17] allow queuing delays and the usage of multiple egress queues per egress port for scheduled traffic in their incremental approach (SMT-INC). Thus, the assignment of streams to traffic classes is not fixed in advance and part of the considered scheduling problem. If a stream cannot be scheduled, backtracking is used. That means that the schedule of an already scheduled stream is unfixed. The unfixed stream and the new stream are scheduled together. This particular SMT approach immediately results in the final solution returned while others first provide a feasible solution which is further improved.

The authors of [17] report that the runtime of SMT-INC grows linearly for up to 100 streams. Pozo et al. [19] refined the incremental approach of SMT-INC to leverage this observation (SMT-DEC). Instead of scheduling all streams with a single run of SMT-INC, they group streams in small subsets and schedule each subset individually with SMT-INC. However, there is no backtracking between streams of different subsets. Thus, SMT-DEC may be a tradeoff between scalability and schedulability for some problem instances.

Gavrilut et al. [15] propose a Greedy Randomized Adaptive Search Procedure (GRASP) heuristic which allows queuing and that uses multiple egress queue per egress port. A greedy randomized algorithm is used to construct initial solutions. It combines multiple heuristics and postprocessing. The found solution is improved by a local search procedure. These steps are repeated until a stopping criterion is met. The details of these heuristics are elaborated in [20]. Resource conflicts are resolved with queuing delay in intermediate nodes instead of later transmissions at talkers. Additionally, the algorithm assigns streams to traffic classes similar to SMT-INC. Thus, a resource conflict between two streams can be resolved by assigning the streams to different traffic classes.

Another heuristic denoted as *Move to Front* (M2F) is presented by Jin et al. [12]. Time is divided into discrete units. The time tabling algorithm iterates for every hop of a frame over all time units and schedules it at the earliest possible time without a resource conflict. However, queuing delay is allowed, i.e., the transmission time at an intermediate node is not determined by the transmission time at the respective talker. In contrast to [15], only a single egress queue per egress port is dedicated to scheduled traffic.

B. Scheduling w/ Joint Routing

Pahlevan et al. [14] present a genetic algorithm for no-wait scheduling and path selection (GenAlg). The heuristic maintains a set of candidate solutions denoted as population. Every candidate solution consists of a mapping from streams to paths which represents the genes of the solution. Biology inspired mechanisms such as crossover, mutation, and selection, are used to construct new solutions. Every candidate solution corresponds to a schedule obtained by time tabling with a fixed frame order, i.e., resource conflicts are reduced by rerouting streams.

In later works, Pahlevan et al. [16] propose a heuristic list scheduler (HLS) to compute schedules and paths. Streams are scheduled incrementally with a time tabling algorithm. HLS computes the resulting flowspan for all possible paths of a stream and selects the path that minimizes the flowspan. That means that resource conflict are reduced by selecting appropriate paths.

Falk et al. [18] and Schweissguth et al. [5] give ILP formulations for the joint routing and scheduling of TT streams. While the ILP (ILP-JR-1) of [18] enforces no-wait scheduling, the ILP (ILP-JR-2) of [5] allows queuing delays. Additionally, ILP-JR-2 preprocesses the network topology to reduce the number of variables and constraints which restricts the solution space.

A novel heuristic approach (ConfGraph) for joint routing and no-wait scheduling is presented by Falk et al. [13]. ConfGraph constructs a conflict graph for a given network topology and set of streams. Each vertex in this graph corresponds to the configuration of a single frame, i.e., the transmission offset at the frame's talker and its path. Two vertices are connected by an edge if and only if the respective configuration results in two frames conflicting on some link. Therefore, an independent set of vertices in the conflict graph corresponds to a valid schedule for all frames covered.

C. Reasoning of the Algorithm Selection

There are more than 100 published works presenting scheduling algorithms for the TAS [2]. Thus, only a few can be selected for an in-depth comparison. We selected the algorithms based on three rationales. First, we wanted the comparison to feature various algorithmic methodologies used in the literature, e.g., ILP and SMT solving, Tabu heuristics, GRASP, or custom heuristics. Typically, the selection of the algorithmic approach has great impact on scalability and schedulability and insights into the appropriateness of an approach can be leveraged by future researchers. Second, an interesting comparison should feature various design decisions that are made by the algorithms' authors. Therefore, we included incremental (e.g., Tabu, GenAlg, SMT-INC, ...) and global approaches (e.g., ILP-NoWait, ILP-JR-$\{1, 2\}$), fixed routing (ILP-NoWait, SMT-DEC, M2F, ...) and joint routing (HLS, ConfGraph, GenAlg, ...) approaches, and algorithms allowing queuing delays (SMT-INC, GRASP, M2F, ...) and no-wait algorithms (ILP-NoWait, Tabu, ConfGraph, ...). Moreover, we believe that a comparison of state-of-the-art approaches should consider multiple seminal works that influenced the field. Tabu and ILP-NoWait from [4] and SMT-INC from [17] are referenced by almost every other work about scheduling for TAS. Additionally, these were the first works using ILP and SMT solving for TAS scheduling [2]. ConfGraph and GRASP are among the very few works about TAS scheduling published in journals. Other algorithms, i.e., GenAlg and HLS, are not seminal for themselves but are the most relevant works with the respective algorithmic methodology.

We do not suggest that the compared works are the most recent works or that the field is dominated by them. In fact, scheduling algorithms for the TAS are a growing field [2, Fig. 19] and new algorithms were proposed in recent conferences and journals, e.g., [21] and [22]. A comprehensive survey of the field up to June 2023 can be found in [2].

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

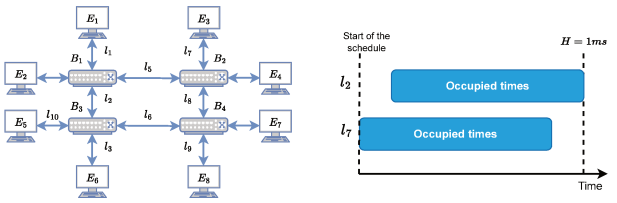
D. Implementation

We implemented all algorithms with the objective function given in the respective papers. All objectives are related to minimizing stream latencies. The scheduling algorithms without routing are combined with Dijkstra's algorithm to compute stream paths. The effort needed to implement the algorithms differs significantly. In particular the descriptions of GenAlg and M2F were rather vague and hard to interpret, which made their implementation challenging. Table 1 reports our subjective perception on implementation effort. We slightly modified some algorithms such that they account for processing and propagation delays in addition to transmission delays. These modifications are needed to make their resulting schedules comparable.

IV. Illustrating Example

We demonstrate the definition of the scheduling problem from Section II-B and the problem variations mentioned in Table 1 by an example. First, we describe the basic setting. Then, we explain the technique of incremental scheduling. Afterwards, we demonstrate how no-wait scheduling restricts the solution space of a scheduling problem. Finally, we show that joint routing algorithms can solve problem instances that are infeasible with fixed paths.

A. Basic Setting



(a) Network topology of the illustrating example.

(b) Times occupied by frame transmissions on l_2 and l_7.

Fig. 2. Illustrating example.

Figure 2(a) depicts a network topology composed of four bridges, eight end stations, and various links. All egress ports have a transmission bandwidth of 100 Mb/s. We assume a set of 101 streams. All streams share the same period $T_s = 1\,\text{ms}$ and deadline 1 ms. Thus, the hyperperiod is $H = 1\,\text{ms}$. 50 streams are sent from E_8 to E_3 and from E_6 to E_1, respectively. A single stream s_{101} is sent from E_5 to E_3. Furthermore, assume that all streams except s_{101} are in the same traffic class and s_{101} is assigned to a different traffic class before scheduling. The paths of the streams are fixed in advance. The path of the streams from E_8 to E_3 is (l_9, l_8, l_7), the path of the streams from E_6 to E_1 is (l_3, l_2, l_1), and the path of s_{101} is (l_{10}, l_2, l_5, l_7).

B. Incremental Scheduling

Incremental scheduling is a scheduling technique. Frames are scheduled one after another and the schedule of previously scheduled frames is considered fixed when scheduling a new frame. For the sake of the example, assume that the frames of all streams except for s_{101} are already scheduled. Figure 2(b) depicts time intervals of the hyperperiod that are occupied by frame transmissions on link l_2 in this case. An incremental scheduler will not rearrange these frame transmissions and can only schedule the frames of s_{101} during times without transmission, i.e., at the beginning of the hyperperiod in this example as depicted in Figure 2(b).

C. Queuing

Some scheduling algorithms do not allow queuing delays. For instance, this restricts the set of possible solutions considerably with incremental schedulers. It is only possible to schedule a frame if there are time gaps without transmissions on all links on the frame's path that match with the transmission delays on the path. In the example, the stream s_{101} can only be scheduled if there are matching time gaps on the links l_{10}, l_2, l_5, and l_7. While l_{10} and l_5 are only used by s_{101}, the links l_2 and l_7 are heavily loaded with other streams. Figure 2(b) show that there are possible transmission times for the frames of s_{101} on l_2 and l_7, but they do not match. Therefore, no schedule for s_{101} can be found. However, schedulers that allow queuing can schedule s_{101} as its frames can wait in the egress port of B_2 until all other frames were transmitted on l_7.

D. Joint Routing

Scheduling algorithms that simultaneously compute the paths for all streams are denoted as joint routing approaches. The discussed problem from the last paragraph cannot be resolved if no queuing is allowed and the stream paths are fixed. However, if the scheduler is allowed to select the stream paths instead of using fixed paths, the situation can easily be resolved by routing s_{101} over (l_{10}, l_6, l_8, l_7). More complicated examples with queuing delays can be constructed. In conclusion, joint routing algorithms can find solutions when no feasible solution exists in a setting with fixed paths.

V. Methodology

We explain the methodology used for performance evaluation. First we introduce the experimental model. Some of its parameters are of random nature. Therefore, we explain how complex parameter studies can be conducted with such parameters such that the studies can be well repeated. To that end, we provide a set of useful problem instances for download. Finally, we describe the execution environment and the framework in which the scheduling algorithms are carried out to produce the results in Section VI.

A. Experimental Model

An experiment comprises a network model and a traffic model including QoS requirements. We study the behavior of the algorithms for different topologies and workloads. Furthermore, special problem instances are provided for testing protection mechanisms and another set of infeasible problem instances are given to study how scheduling algorithms behave under challenging conditions.

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

1) Network Model: We use five different topologies for the interconnection of bridges. Line and star topologies are common in classical Ethernet networks. Ring topologies are important in factory automation use cases of TSN [13][18]. Scale-free networks (SFNs) are random graphs with a power law distribution of node degrees, i.e., they feature a few hubs with high node degrees and many nodes with low node degrees. They can be used to model hierarchical computer networks. Random regular graphs (RRGs) are random graphs in which all nodes have the same node degree. They resemble switching networks in which all switches have the same number of ports. We construct RRGs with node degree 4.

The nodes in the topologies represent bridges. Every bridge is connected to four end stations. Bridges have a processing delay of 1 µs. All links are full-duplex and modelled by two unidirectional links. Each unidirectional link has an egress port at its head end and an ingress port at its tail end, and all egress ports are equipped with eight egress queues. Each link has a propagation delay of 100 ns and the respective egress port has a transmission capacity of 1 Gb/s.

2) Traffic Model: The traffic model consists of a set of streams with latency and deadline requirements. Talkers and listeners of a stream are chosen randomly from the set of end stations. Every stream may have one or multiple listeners. Streams periodically send a fixed number of frames as a burst. Frames sizes on the physical layer are in the range $[84\,\mathrm{B}, 1542\,\mathrm{B}]$. The period T_s and the frame size f_s of a stream s is either the same for all streams or drawn from a set given in Table 2. The period sets $\{1\,\mathrm{ms}\}$ and $\{1\,\mathrm{ms}, 2\,\mathrm{ms}\}$ correspond to industrial use cases of isochronous traffic [8]. The period set $\{20\,\mathrm{ms}, 50\,\mathrm{ms}, 100\,\mathrm{ms}\}$ represents typical periods of sending tasks in automotive scenarios [9]. Use cases in the aerospace domain are constructed with the period set $\{2\,\mathrm{ms}, 16\,\mathrm{ms}, 128\,\mathrm{ms}\}$ [10]. According to [9] and [10], the first two period sets are also representative for automotive and aerospace use cases. We the period set $\{500\,\mu\mathrm{s}\}$ to represent factory automation use cases that require extremly low latencies and short cycle times. The number of frame instances is the overall number of frame transmissions, i.e., the number of frames sent from all talkers multiplied by the lengths of their individual paths. Talkers and listeners are generated such that the number of frame instances equals a value in Table 2. These numbers of frame instances correspond to about 20 – 1300 streams per problem instance which covers the range of stream numbers in realistic use cases [2, Table 6 and 7]. Moreover, random frame sizes are assigned to streams such that the overall traffic load for all streams is the same as for constant frames sizes with $\frac{1542+84}{2} = 813$ bytes.

Every stream has a deadline and a maximum latency requirement. In the literature, the periods of a all streams are synchronized in the sense that they start from a common time t_0. The earliest transmission time of a frame of stream s in period k is at $t_0 + k \cdot T_s$ and the frame has to be received by all its listeners at $t_0 + (k+1) \cdot T_s$ which is denoted as deadline[2]. Thus, we assume that all deadlines are before the end of a period which is common in industrial use cases [8, Table 3] and the literature for TAS scheduling [2, Section V.C]. We define the latency of a frame as the difference between the time the last bit of the frame arrives at all of its listeners and the time the first bit of the frame is sent at its talker. The latency of a stream is defined as the maximum of the latencies of the stream's frames during a hyperperiod. Thus, latencies and deadlines differ in the reference time. While deadlines are relative to the respective stream's period, latencies are relative to the transmission start at the respective stream's talker. The maximum allowed latency is given by a parameter in Table 2.

[2]Other definitions of earliest transmission time and deadlines are possible but not common in literature.

Parameter	Possible values	# instances
#Bridges	10, **20***, 50, 100	160
#Frame instances	250, 500, **1000***, 2000, 4000, 8000	240
Topology	Line, **ring***, star, **RRG***, scale-free	100
#Frames/period	**1***, 2, 4	120
Frame size f_s	**Random***, 84 B, 813 B, 1542 B	120
Stream periods T_s	$\{500\,\mu\mathrm{s}\}$, $\{1\,\mathrm{ms}\}$, $\{\mathbf{1\,ms, 2\,ms}\}$*, $\{20\,\mathrm{ms}, 50\,\mathrm{ms}, 100\,\mathrm{ms}\}$, $\{2\,\mathrm{ms}, 16\,\mathrm{ms}, 128\,\mathrm{ms}\}$	160
Latency	0.25, 0.5, **1*** × the respective stream's period	120
#Listeners/stream	**1*** (unicast), 2, 4, 8, 16	200

TABLE 2
POSSIBLE AND DEFAULT VALUES FOR THE PROPOSED PARAMETER STUDIES. DEFAULT VALUES ARE **BOLD** AND INDICATED BY "*".

B. Repeatable Parameter Studies

The construction of an experiment may be characterized by multiple parameters. Parameters are either deterministic or random. Each deterministic parameter is configured to a specific value. For instance, the number of bridges in a network topology is a deterministic parameter and a topology with exactly 10 bridges is constructed when the parameter is set to 10. Random parameter are configured to be drawn from some distribution. For instance, the period of a stream is a random parameter and it may be configured to be drawn from the uniform discrete distribution with the possible realisations $\{1\,\mathrm{ms}, 2\,\mathrm{ms}\}$ when a stream is constructed.

A parameter study for a parameter p consists of sets of 20 experiments per parameter configuration. All parameters except for p are configured to their respective default value in all of these sets. If p is a deterministic parameter, each of these sets is constructed with p configured to another of its possible values. If p is a random parameter, each of these sets is constructed with p configured to another distribution of its possible distributions. We reuse the same realisations of the random parameters for different sets whenever it is possible to do so. For example, the same random frame sizes are used in the i-th, $0 \leq i < 20$, experiment in every set of the parameter study of the maximum latency parameter. This methodology increases the comparability of evaluation results for different values of p as random noise is minimized and the overall traffic amount is equal in the experiments of different sets.

To make the performance studies in this paper repeatable, we provide series of $m = 20$ samples for every set of every

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

parameter study for download[3].

C. Problem Instances for Evaluation of TSN Scheduling Algorithms

The experiment model in Section V-A has the following random parameters: topologies for RRGs and SFNs, talkers and listeners of streams, stream periods, and a stream's frame size. To make experiment repeatable, we provide series of them. Parameter values for parameter studies and the default values of these parameters are compiled in Table 2.

We further include a set of infeasible problem instances with RRG and ring topologies. Each of them contains 100 streams with a subset of ten streams for which no valid schedule exists, but every subset of nine streams or less can be scheduled. These unfeasible problem instances are useful to test the ability of scheduling algorithms to find schedules under challenging conditions and to decide whether a given problem is unfeasible. Additionally, these instances can be employed to evaluate the adaption of incremental heuristics to online scenarios. In such scenarios, streams must be integrated into an existing schedule if possible. If it is not possible, the current schedule is discarded and all streams are rescheduled again incrementally. The streams in infeasible instances can be added one after another to simulate an online scenario. The construction of the infeasible instances guarantees that all streams are rescheduled after at most 100 added streams. Thus, online scenarios can be tested for an indefinite number of streams with these instances.

Some of the above and the following series of parameter samples are provided in the online resources without being utilized in this paper. To test Frame Replication and Elimination for Reliability (FRER), we include 2-connected RRG and ring topologies which remain connected in case of any single link failure. Likewise, series of streams with multiples listeners are added to support multicast studies.

D. Execution Environment

We perform all evaluations on an Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz running Linux. Every algorithm is executed with a single thread and 4 GB RAM if not stated otherwise. We use CPU pinning to guarantee that every computation runs exclusively on a specific CPU core. ILP models are solved with Gurobi 10 [23]. SMT models are solved with Z3 4.12.1 [24]. Each computation is configured to timeout after 1 h per problem instance.

E. Evaluation Framework

We implemented an object-oriented evaluation framework. The framework is structured into packages for instance generation, path computation, scheduling, evaluation runtime, and result output. Besides of the implementation of scheduling and path selection algorithms, it features an evaluation pipeline, executes and measures experiments parallely in a thread pool, and reports results in a human-readable HTML format for debugging. The core of the framework is the construction of parameter studies. Constructing and composing experiments for a large number of very different studies is hard. To mitigate this problem, we use the `FactoryMethod` [25] pattern for generating scheduler objects, topologies, streams, links, devices, traffic models, and period models. Multiple implementations of this pattern are provided for every category of generated objects. This modular approach allows to construct parameter studies by plugging together implementations of the factory method pattern. The methodology for constructing problem instances from Section V-B is implemented by a hard-coded composition of these factory implementations. Thus, problem instance generation is just a special case of our more general framework and new parameter studies are simple to implement. Additionally, new parameter distributions or completely new parameters can be added to the framework by implementing a single interface.

[3] [redacted]

VI. Evaluation

In this section we compare the performance of the 11 scheduling algorithms from Section III. After comparing the computation time of the algorithms, we demonstrate that the algorithms differ in the size of the problem instances they can solve. Then, we compare the algorithms with respect to the length of the resulting GCLs. We investigate to what extent multi-threading can reduce solving times. We study the algorithms' ability to find solutions under challenging conditions. Finally, we compare the schedule quality in terms of end-to-end latency of transmitted frames. We computed 95% confidence intervals for all average values, but omitted them in figures and tables as they were smaller than 5% of the calculated average values.

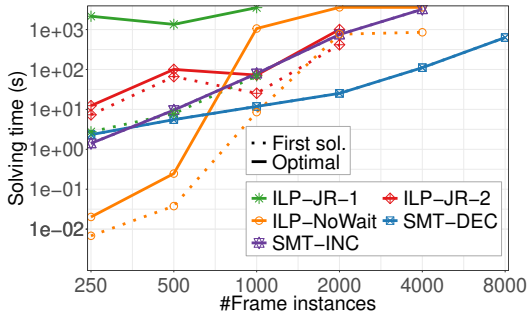
A. Solving Time

We compare solving times of the scheduling algorithms in various parameter studies from Table 2.

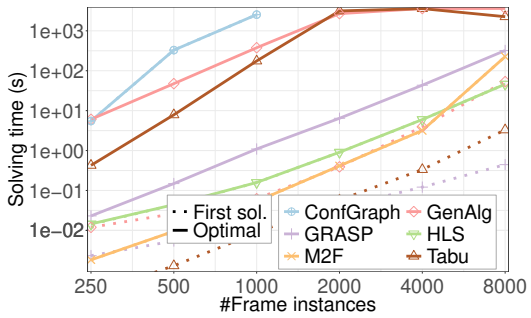
1) Varying the Number of Frames: We compute schedules on problem instances with various number of frame instances and report the times needed to find a first solution and a final solution. The time to compute a first solution may be important as some applications require frequent computation of new schedules that do not need to be optimal. We stopped the computation after a timeout of 1 h if the computation was not finished, yet. Figures 3(a)–3(b) report the results for exact solutions and for heuristic solutions averaged over ring and RRG topologies. We observe that more frame instances lead to longer computation times for all algorithms. More frame instances increase the solution space and make it harder to find valid solutions as more frame transmissions compete for transmission resources and cause more resource conflicts.

As expected, exact approaches lead to longer solving times compared to heuristic algorithms. For problem instances with at most 1000 frame instances, the computation time for ILP-JR-1 is an order of magnitude higher than the computation of ILP-JR-2, i.e., its improvements are effective. Remarkably, ILP-JR-2 is even faster than ILP-NoWait for 1000 frame instances despite ILP-JR-2 is a more complex joint routing

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503



(a) Exact approaches.



(b) Heuristic approaches.

Fig. 3. Solving times for first and optimal solutions for varying numbers of frame instances.

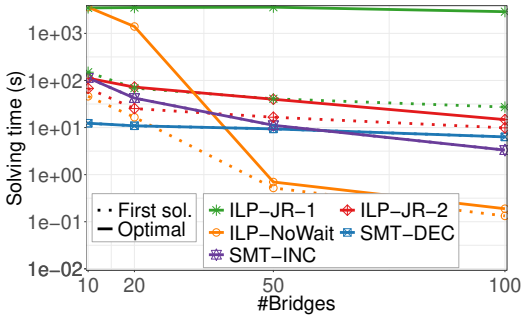


Fig. 4. Solving times for varying numbers of bridges in ring topologies.

model. This is due to ILP-JR-2 using a simpler objective function than ILP-NoWait which allows faster termination. However, ILP-NoWait finds its first solution earlier than ILP-JR-2. With ILPs, first solutions can be found faster, but the needed time also quickly exceeds 1 h. In contrast, SMT-INC and SMT-DEC yield only a final solution. SMT-DEC is by far the fastest exact approach, i.e., its modification of SMT-INC is effective. HLS, M2F, and GRASP are very fast even for 8000 frames while GenAlg, and Tabu are stopped after 1 h on problem instances with 2000 or more frames. ConfGraph and GenAlg are even slower than the exact approaches SMT-INC. They are so slow because they compute both routing and scheduling. In contrast, Tabu is able to quickly find an initial solution even for 8000 frame instances, but takes long time to improve results.

2) Varying the Topology Size: When varying the number of bridges, the number of frame instances is set to 1000. Figure 4 depicts the results for exact solutions in ring topologies only as ILP-JR-$\{1, 2\}$ were not able to schedule a single RRG topology with 1000 frame instances. For brevity, we omit the very similar figure for heuristic approaches. Larger topologies result in shorter solving times. This is due to the constant number of frame instances competing for an increasing amount of transmission bandwidth. Thus, both the number of frames and the number of bridges contribute to the complexity of a problem instance.

3) Varying the Solution Quality: ILP solvers are capable of inferring optimality gaps. That means they can give upper bounds for the absolute and relative difference between the objective values of the incumbent solution and the optimal solution despite the optimal solutions is unknown during solving. The ILP-based approaches have a systematic disadvantage compared to heuristic approaches in the evaluation of solving times. The latter will simply terminate when no better solution is available while ILP approaches may work hours for the last 0.1% to optimality. Therefore, we reevaluate the ILP approaches ILP-NoWait, ILP-JR-$\{1, 2\}$ with a configured ILP solver that terminates when the relative optimality gap is less than 5%. Table 4 presents solving times for the default parameters, i.e., 20 bridges and 1000 frame instances. The results reveal that ILP-NoWait spends 50% of the solving time for the last 5% gap to optimality. However, this is not the case for ILP-JR-$\{1, 2\}$. Thus, they spent most of the solving time to find an acceptable solution. We remark that it may be the case that the optimal solution is found early during solving, but inferring the optimality gap takes most of the reported solving time. However, this is not the case in the presented evaluations.

Algorithm	ILP-NoWait	ILP-JR-1	ILP-JR-2
Default (s)	574	1990	63.4
5% gap (s)	268	1821	45.4
Rel. (%)	46.7	91.5	71.5

TABLE 4
SOLVING TIMES W/ AND W/O AN OPTIMALITY GAP OF 5%.

B. Scalability

TSN scheduling is known to be NP-complete [4]. Therefore, algorithms may be unable to solve larger problem instances. We investigate this issue for varying number of frames and bridges in ring and RRG topologies within a limit of 1 h computation time and 4 GB RAM. Table 3 compiles the fraction of solved problem instances (out of 20) for various parameter configurations.

Heuristic approaches except for ConfGraph can schedule most considered problem instances within the given time and memory constraints. An exception is ConfGraph which builds a large conflict graph to find solutions, which scales badly. We observe that ring topologies are more difficult to solve than RRG topologies for most algorithms, especially for heuristics. In rings, frames compete for a small number of links, leading to more resource conflicts and a larger solution space for ILP and SMT solvers. In contrast, some methods with joint routing and scheduling can solve ring topologies better than RRG topologies. This is because meshed networks have more potential paths that need to be explored, which adds complexity. As a result, ILP-JR-{1,2} are not able to

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

Algorithm	Method	Topo.	# frame instances						# bridges			
			250	500	1000	2000	4000	8000	10	20	50	100
M2F	Heuristic	Ring	100	100	100	100	100	100	100	100	100	100
		RRG	100	100	100	100	100	100	100	100	100	100
Tabu	Heuristic	Ring	100	100	100	100	100	25; F	100	100	100	100
		RRG	100	100	100	100	100	100	100	100	100	100
ConfGraph	Heuristic	Ring	100	100	0; M	0; M	0; M	0; M	0; M	0; TM	100	95; T
		RRG	100	100	65; TM	0; M	0; M	0; M	0; TM	65; TM	100	100
GenAlg	Heuristic	Ring	100	100	100	100	100	100	100	100	100	100
		RRG	100	100	100	100	100	100	100	100	100	100
GRASP	Heuristic	Ring	100	100	100	100	100	100	100	100	100	100
		RRG	100	100	100	100	100	100	100	100	100	100
HLS	Heuristic	Ring	100	100	100	100	100	100	100	100	100	100
		RRG	100	100	100	100	100	100	100	100	100	100
SMT-INC	SMT	Ring	100	100	100	100	30; T	0; T	100	100	100	100
		RRG	100	100	100	100	0; T	0; T	100	100	100	100
SMT-DEC	SMT	Ring	100	100	100	100	100	100	100	100	100	100
		RRG	100	100	100	100	100	100	100	100	100	100
ILP-JR-1	ILP	Ring	100	100	100	0; M	0; M	0; M	100	100	100	100
		RRG	100	30; M	0; M	0; M	0; M	0; M	0; M	0; M	0; M	0; M
ILP-JR-2	ILP	Ring	100	100	100	100	0; M	0; M	100	100	100	100
		RRG	100	90; M	0; M	0; M	0; M	0; M	0; M	0; M	0; M	0; M
ILP-NoWait	ILP	Ring	100	100	100	95; T	0; T	0; TM	100	100	100	100
		RRG	100	100	100	100	75; T	0; T	100	100	100	100

TABLE 3

FRACTION (%) OF SOLVED PROBLEM INSTANCES. PARAMETER CONFIGURATIONS FOR WHICH AN ALGORITHM TIMED OUT OR RAN OUT OF MEMORY ARE INDICATED WITH "T" OR "M", RESPECTIVELY. AN "F" INDICATES THAT A HEURISTIC FAILED TO FIND AN INITIAL SOLUTION.

solve RRG topologies with 1000 frame instances while they can solve corresponding ring topologies. SMT-INC and SMT-DEC outperformed the ILP-based approaches with respect to solving times. However, this observation is only specific for the studied algorithms and cannot be generalized. Additionally, ILP approaches tend to run out of memory in contrast to SMT algorithms. The only exact algorithm that is able to schedule all problem instances with 8000 frame instances is SMT-DEC. This is due to its approach of scheduling small subsets of frames incrementally with SMT-INC instead of scheduling all frames incrementally in one run of SMT-INC.

C. GCL Length

The maximum length of the GCLs is limited in hardware bridges and many gate events may result in a waste of bandwidth due to guard bands. Thus, the lengths of the GCLs resulting from scheduling are important. Most end stations are talker or listener of only a few TT stream in the constructed problem instances which results in rather short GCLs on average. However, the GCLs of highly loaded egress ports of bridges are more interesting as their construction is non-trivial. Thus, we consider only egress ports that connect bridges with each other to remove the mentioned bias. We report average GCL lengths for the topology parameter study in Table 5. We observe the trend that highly meshed topologies require less GCL entries per egress port on average. This is due to a higher number of egress ports for an equal number of frame instances. All algorithms except for M2F and SMT-DEC result in similar GCL lengths. GRASP and SMT-INC do not use more GCL entries than the other approaches despite their assignment of frames to different traffic classes. The shortest GCLs are constructed by M2F and SMT-DEC. M2F iterates over time instants and schedules frame transmissions at the first suitable time found, resulting in back-to-back frame transmissions and thus less required GCL entries. SMT-DEC schedules small subsets of frames together in short time intervals, which also results in back-to-back frame transmissions.

Algorithm	Line	Ring	Star	RRG	Scale-free
M2F	17.9	15.7	11.6	7.75	9.95
Tabu	24.5	21	13.4	9.18	11.9
ConfGraph	23.8	-	15.9	11.9	-
GenAlg	22.7	20.2	-	15.4	-
GRASP	22.7	20.6	15.2	10.1	12.9
HLS	22.9	20.2	13.1	10.2	13.2
SMT-INC	23.5	20.3	13.3	9.02	11.7
SMT-DEC	16.5	15	11.3	8.22	9.55
ILP-JR-1	21.7	17.9	12.9	-	13.3
ILP-JR-2	21.7	19.1	13.4	-	12.4
ILP-NoWait	22.4	19.4	13.7	9.11	12

TABLE 5

AVERAGE GCL LENGTHS FOR DIFFERENT TOPOLOGIES. A "-" INDICATES THAT THE RESPECTIVE ALGORITHM WAS NOT ABLE TO SCHEDULE A SINGLE PROBLEM INSTANCE WITH THE RESPECTIVE TOPOLOGY.

D. Multi-Threading in TSN Scheduling

Computations may be accelerated by multi-threading on many cores. Prerequisite is that programs support such multi-threading. The existing heuristics are not multi-threaded, but they may be extended for that purpose, which requires additional effort with uncertain gain. In contrast, SMT and ILP solvers support multi-threading. On the one hand, a problem can be solved on many cores in parallel. On the other hand, multi-threading imposes inter-thread synchronization and communication overhead. Therefore, the benefit of parallelization of TSN scheduling is unclear and it may depend on the specific model.

We investigate this issue by solving the SMT and ILP models with up to 8 pinned CPU cores and allowing 1 h and 4 GB RAM per thread. We study problem instances with default values and measure the solving time with many threads relative to the one with a single thread. The results are compiled in Table 6.

The SMT-based scheduling model hardly benefits from multi-threading. In contrast, the solving time for ILP-based scheduling models can be clearly reduced in most cases, but the speed-up depends on the algorithm and the specific problem, i.e., ring or RRG topology. We further observe that

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

Algo.	Topo.	Single thread (s)	Threads		
			2	4	8
SMT-INC	Ring	42.3	89%	89.6%	88.9%
	RRG	116	90.5%	89.9%	90.7%
SMT-DEC	Ring	10.9	99.3%	98.7%	99.1%
	RRG	12.8	99.1%	99.7%	99.2%
ILP-JR-1	Ring	3541	102%	80.7%	71.2%
	RRG	-	-	-	-
ILP-JR-2	Ring	72.3	67.2%	69.4%	68%
	RRG	-	-	-	-
ILP-NoWait	Ring	1400	51.9%	36.3%	27.3%
	RRG	723	85.8%	78.5%	62.3%

TABLE 6
SOLVING TIMES W/ MULTIPLE THREADS RELATIVE TO THE SOLVING TIME W/ A SINGLE THREAD. A "-" INDICATES THAT NOT A SINGLE PROBLEM INSTANCE WAS SOLVED DUE TO MEMORY LIMITATIONS.

solving time scales worse than $\frac{1}{n}$ when n is the number of threads, which is due to synchronization overhead and others. Due to this overhead, it is even possible that solving time is extended as in the case of ILP-JR-1.

E. Scheduling Under Challenging Conditions

To construct challenging conditions, we consider the infeasible problem instances where 99 streams can be scheduled but not 100. We study how many of their streams can be scheduled by the different algorithms. For that purpose, we take the first k streams of the 100 lists of stream positions and test whether the algorithms can solve the resulting problem instance. We denote the largest k which is solvable as the number of schedulable streams.

The exact algorithm ILP-NoWait can schedule 99 streams for both ring and RRG topologies. The other exact algorithms ILP-JR-$\{1,2\}$ can schedule 99 streams for RRG topologies but are not able to solve the problem instances with many streams within 1 h so that the experiment could not be completed. However, if completed, 99 streams would be scheduled.

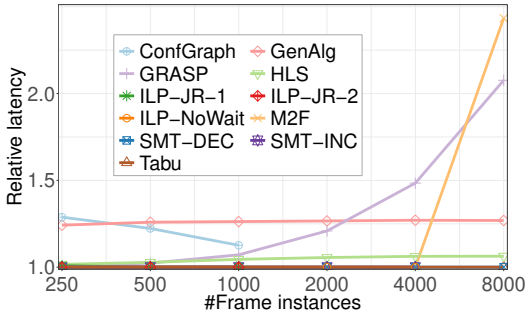
This is different with heuristics. Table 7 compiles their average number of schedulable streams which is between 56.5 and 93.7. Tabu, GRASP, and HLS can schedule many streams in RRG topologies (85.0 – 93.7). Others, M2F and GenAlg, accommodate only 66.6–79.1 streams. Thus, while heuristics are faster and can solve larger problem instances, they cannot find solutions for difficult problem instances although solutions exist. Moreover, the considered heuristics significantly differ in their ability to find solutions. Similar to the heuristics, SMT-DEC is unable to schedule most streams and performs worse than the heuristics on ring topologies. This is due to the incremental design of SMT-DEC that favors scalability over schedulability.

Topo.	M2F	Tabu	GenAlg	GRASP	HLS	SMT-DEC	ILP-NoWait
Ring	66.66	90.14	66.86	79.34	56.50	25.00	99
RRG	79.10	91.93	83.42	85.03	93.73	80.15	99

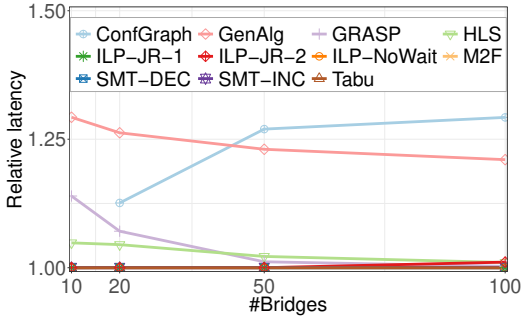
TABLE 7
MAXIMUM NUMBER OF SCHEDULED STREAMS UNDER CHALLENGING CONDITIONS IN PROBLEM INSTANCES WITH 99 STREAMS. CONFGRAPH AND ILP-JR-{1,2} ARE NOT LISTED AS THEY CANNOT SOLVE INSTANCES WITH 100 STREAMS WITHIN 1 H.

F. Latency

All evaluated algorithms were designed by their respective authors to construct schedules with low end-to-end latencies for frames as most real-time applications require ultra-low latencies. Therefore, it is reasonable to compare the algorithms with respect to this quality measure. However, we remark that isochronous traffic only features deadline requirements [8] and minimizing the stream latencies is just an objective to compare schedules. The theoretical minimum delay of a frame is the sum of processing, propagation, and transmission delays along the shortest path from its talker to its listener. We calculate the achieved frame latency relative to that theoretic minimum and call it relative latency. We report relative latencies averaged over all frames and runs in Figures 5(a)–5(b). In the following, we explain them based on the properties of the algorithms (cf. Table 1).



(a) Varying the number of frame instances.



(b) Varying the number of bridges.

Fig. 5. The average latencies relative to the latency on the shortest path w/o queuing delay.

GRASP and MF resolve resource conflicts by introducing queuing delays, which explains their increased latencies for many frames. However, in contrast to GRASP, MF suffers from increased latencies only with very large problem instances. HLS, GenAlg, and ConfGraph implement joint routing and scheduling, i.e., they resolve resource conflicts by changing the paths of frames, which also result in longer latencies than needed. Tabu and ILP-NoWait neither allow queuing nor rerouting and, therefore, achieve minimum relative latencies. These observations also hold for the exact approaches ILP-JR-$\{1,2\}$. Thus, they do not seek sufficiently for solutions with low latency.

Remarkably, SMT-INC and SMT-DEC do not introduce

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

queuing delays despite the fact that their modelling does not enforce a no-wait constraint and no objective function is used. They implement an isolation constraint which forbids frames from different streams to reside in the same queue at the same time. Thus, it is beneficial to schedule frames without queuing delays as more frames can be scheduled in a given planning horizon.

When considering a varying number of bridges, we observe very similar results. Contrary to other approaches, ConfGraph results in higher latencies for larger topologies as it resolved resource conflicts primarily through path selection.

VII. Discussion

We highlight findings from the presented performance comparison and give guidelines for selecting an algorithm based on the evaluation results.

A. Summary

Existing scheduling algorithms for TSN scheduled traffic differ a lot. Most exact approaches based on SMT or ILP tend to solve only smaller problem instances. Some fail on medium-size problem instances due to limited computing time (SMT-INC, ILP-NoWait), others due to limited memory (ILP-JR-{1,2}). However, SMT-DEC scales well even for large problem instances with 8000 frame instances. Also the ConfGraph heuristic can solve only small problem instances. The other heuristics GenAlg, GRASP, HLS, M2F, and Tabu find a first valid schedule much faster than exact approaches and most of them can tackle larger problem instances.

The considered heuristics are not designed such that they can profit from multi-threading on many cores. In contrast, SMT and ILP models can be accelerated at least in theory. However, we achieved a substantial speedup only for ILP-NoWait.

When exact approaches terminate successfully, they always find a solution of a problem instance if one exists. We showed that this does not hold for heuristic approaches. They are often unable to find existing solutions under challenging conditions, which is a significant drawback. In practice that means they can schedule fewer streams although more are schedulable. Among the evaluated algorithms, only Tabu was able to admit more than 90% of the streams for lowly and highly meshed topologies.

The schedule quality in terms of stream latency, the common optimization goal of the investigated algorithms, differs a lot. Algorithms that allow queuing (GRASP, M2F) or rerouting (HLS, GenAlg, ConfGraph) to find a valid schedule lead to longer stream latencies than algorithms that do not allow these features (Tabu, ILP-NoWait, ...).

We remark that scheduling algorithms cannot be compared by a single metric and no conclusion can be drawn from a single experiment. For instance, some use cases require really fast schedule computation while other use cases require small latencies but computation times are irrelevant. Thus, a careful evaluation of the use case is needed before selecting an algorithm. However, ConfGraph and GenAlg performed badly with respect to both solving times and latencies while SMT-DEC was the fastest algorithm that resulted in minimum latencies. The presented evaluations represent a starting point to assess the current state-of-the-art and to identify promising methodologies for future research.

B. Conclusions & Guideline

The evaluation results indicate various lessons relevant for practitioners that must select an algorithm. We formulate selection guidelines based on these observations.

The number of available GCL entries in hardware bridges is still limited. These entries are not only used to implement TAS schedules, but to protect different traffic classes from each other, e.g., AVB and BE traffic. Thus, GCL entries are an expensive resource in environments with legacy devices or different traffic classes. We recommend M2F and SMT-DEC in such cases as they result in few GCL entries required to implement the TAS schedule. Applications that require ultra-low latencies should not employ joint routing algorithms, e.g., ConfGraph, GenAlg, HLS, ILP-JR-{1,2}, as these algorithms resolve resource conflicts by selecting suboptimal paths. In contrast, ILP-NoWait and Tabu result in minimal latencies without drawbacks with respect to schedulability across all experiments. Some applications of TSN, such as in-vehicle communication, require frequent recomputation of schedules. The evaluations showed that most algorithms find valid solutions fast but take significantly more time to optimize some objective function. We recommend Tabu, HLS, and GRASP in these cases as they construct feasible solutions several orders of magnitude faster than other algorithms. Heuristic algorithms fail to produce valid schedules under challenging conditions such as tight deadlines combined with heavily loaded links. They are not able to find valid schedules in these cases although a schedule exists. Therefore, ILP-based approaches such as ILP-NoWait are recommended to be used under such conditions. Additionally, these algorithms can prove the non-existence of a valid schedule. Finally, costs for software packages and developers can be substantial when implementing a scheduling algorithm. Especially ILP solvers are expensive and thus not available for many practitioners. The compared algorithms differ significantly in required implementation effort. Based on the authors experience, we recommend Tabu, HLS, SMT-INC, and SMT-DEC for fast and efficient implementation if implementation costs are a limiting factor.

VIII. Conclusion

In this paper, we proposed a set of problem instances that was designed to make parameter studies comparable across different schedulers. This set is released to the public to allow future works a comparison with the current state-of-the-art. To that end, we selected and implemented 11 highly influential algorithms with different methodologies and reported evaluation results for the proposed problem instances. First, we confirmed that heuristic algorithms are faster than exact approaches and that scheduling scales badly with the number of frame instances. The numerical results of this comparison can be used as a reference in future works that propose

This article has been accepted for publication in IEEE Transactions on Industrial Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TII.2024.3385503

new scheduling approaches. Second, we reported the counter-intuitive finding that scheduling in larger topologies may be simpler than in smaller ones. We demonstrated that exact joint routing approaches are not able to schedule medium-sized problem instances, an observation that does not hold for heuristic joint routing approaches. We showed that M2F and SMT-DEC result in significantly smaller GCLs than other algorithms. Then, we demonstrated that some exact approaches (ILP-NoWait, ILP-JR-2) benefit heavily from multi-threading, but this observation depends on the modelling and the topology. However, the speedup of using n parallel threads is substantially less than $\frac{1}{n}$. Additionally, we evaluated the capabilities of heuristic algorithms under challenging conditions, i.e., when no feasible schedule exists. We observed that Tabu was able to admit more than 90% of the streams while other heuristics struggled with these instances. Finally, we compared schedule quality with respect to stream latencies. We observed significantly higher latencies than necessary due to queuing delays for GRASP and M2F, and suboptimal routings for GenAlg and ConfGraph.

In summary, the evaluations quantified the capabilities of current state-of-the-art solvers. Future works may explore novel metrics for measuring schedule quality or add new algorithms to the comparison. The observed results show that the solvers for exact methods and modeling methodologies of heuristics have significant implications for runtime, schedule quality and properties like number of GCL entries. For instance, GenAlg and ConfGraph were slow and resulted in high latencies. In contrast, HLS and Tabu were remarkably simple to implement and represent a good tradeoff between solving times and latencies. The enhanced incremental scheduler SMT-DEC was almost as fast as some heuristics, i.e., its methodology is superior to SMT-INC and current ILP approaches. We do not recommend to implement the joint routing ILPs ILP-JR-$\{1, 2\}$ as they scaled badly and were not able to schedule medium-sized instances.

We conclude that there is no all-in-one methodology for schedule computation in TSN for arbitrary use cases. Future research in scheduling may explore different solving and modeling methods for TSN scheduling to achieve the best tradeoff between performance and schedule quality for a specific use-case.

References

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

[2] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN)," *IEEE Access*, vol. 11, pp. 61 192–61 233, 2023.

[3] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, *et al.*, "Performance Comparison of IEEE 802.1 TSN Time Aware Shaper (TAS) and Asynchronous Traffic Shaper (ATS)," *IEEE Access*, vol. 7, pp. 44 165–44 181, 2019.

[4] F. Dürr *et al.*, "No-wait Packet Scheduling for IEEE Time-Sensitive Networks (TSN)," in *RTNS*, 2016.

[5] E. Schweissguth *et al.*, "ILP-Based Routing and Scheduling of Multicast Realtime Traffic in Time-Sensitive Networks," in *IEEE RTCSA*, 2020.

[6] A. C. T. dos Santos, B. Schneider, and V. Nigam, "TSNSCHED: Automated Schedule Generation for Time Sensitive Networking," in *Formal Methods in Computer Aided Design (FMCAD)*, 2019.

[7] E. Schweissguth, H. Parzyjegla, P. Danielis, *et al.*, "TSN Scheduler Benchmarking," in *IEEE International Conference on Factory Communication Systems (WFCS)*, 2023, pp. 1–8.

[8] Industrial Internet Consortium, *Time Sensitive Networks for Flexible Manufacturing Testbed - Description of Converged Traffic Types*, [Online; accessed 21-September-2023], 2018. [Online]. Available: `https://www.iiconsortium.org/pdf/IIC_TSN_Testbed_Char_Mapping_of_Converged_Traffic_Types_Whitepaper_20180328.pdf`.

[9] S. Kramer, D. Ziegenbein, and A. Hamann, "Real World Automotive Benchmarks for Free," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2015.

[10] M. Boyer, L. Santinelli, N. Navet, J. Migge, and M. Fumey, "Integrating End-System Frame Scheduling for More Accurate AFDX Timing Analysis," in *Embedded Real Time Software and Systems (ERTS)*, 2014.

[11] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, 2020.

[12] X. Jin *et al.*, "Real-Time Scheduling of Massive Data in Time Sensitive Networks With a Limited Number of Schedule Entries," *IEEE Access*, vol. 8, 2020.

[13] J. Falk *et al.*, "Time-Triggered Traffic Planning for Data Networks with Conflict Graphs," in *IEEE RTAS*, 2020.

[14] M. Pahlevan *et al.*, "Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks," in *IEEE ETFA*, 2018.

[15] V. Gavriluţ *et al.*, "AVB-Aware Routing and Scheduling of Time-Triggered Traffic for TSN," *IEEE Access*, vol. 6, 2018.

[16] M. Pahlevan *et al.*, "Heuristic List Scheduler for Time Triggered Traffic in Time Sensitive Networks," *ACM SIGBED Review*, vol. 16, no. 1, 2019.

[17] S. S. Craciunas *et al.*, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," in *RTNS*, 2016.

[18] J. Falk *et al.*, "Exploring Practical Limitations of Joint Routing and Scheduling for TSN with ILP," in *IEEE RTCSA*, 2018.

[19] F. Pozo, W. Steiner, G. Rodriguez-Navas, and H. Hansson, "A decomposition approach for SMT-based schedule synthesis for time-triggered networks," in *IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, 2015.

[20] M. L. Raagaard *et al.*, "Optimization algorithms for the scheduling of IEEE 802.1 Time-Sensitive Networking (TSN)," Technical University of Denmark, Tech. Rep., 2017.

[21] X. Zhou, F. He, L. Zhao, and E. Li, "Hybrid Scheduling of Tasks and Messages for TSN-Based Avionics Systems," *IEEE Transactions on Industrial Informatics*, vol. 20, pp. 1081–1092, 2024.

[22] P.-J. Chaine, M. Boyer, C. Pagetti, and F. Wartel, "Egress-TT Configurations for TSN Networks," ser. RTNS '22, 2022.

[23] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2021. [Online]. Available: `https://www.gurobi.com`.

[24] L. De Moura *et al.*, "Z3: An Efficient SMT Solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[25] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.

1.4 Efficiency of BIER Multicast in Large Networks

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

Efficiency of BIER Multicast in Large Networks

Daniel Merling*, Thomas Stüber*, Michael Menth

Chair of Communication Networks, University of Tuebingen, Germany
{daniel.merling, thomas.stueber, menth}@uni-tuebingen.de

***Abstract*—Bit Index Explicit Replication (BIER) has been introduced by the IETF to transport IP multicast (IPMC) traffic within a BIER domain. Its advantage over IPMC is improved scalability regarding the number of multicast groups. However, scaling BIER to large networks is a challenge. To that end, receivers of a BIER domain are assigned to smaller subdomains. To deliver an IPMC packet over a BIER domain, a copy is sent to any subdomain with a receiver for that packet. Consequently, some links may carry multiple copies of the same IPMC packet, which contradicts the multicast idea.**

In this paper, we propose and compare various algorithms to select subdomains for BIER in order to keep the overall BIER traffic low despite multiple packet copies. We apply them to investigate the traffic savings potential of IPMC and BIER relative to unicast under various conditions. We show that the traffic savings depend on network topology, network size, and the size of the multicast groups. Also the extra traffic caused by BIER depends on these factors. In spite of some redundant packets, BIER can efficiently reduce the overall traffic in most network topologies. Similarly to IPMC, BIER also avoids heavily loaded links. Finally, we demonstrate that BIER subdomains optimized for failure-free conditions do not cause extensive overload in case of single link failures.

***Index Terms*—Bit Index Explicit Replication (BIER), multicast, IP networks, performance evaluation, optimization**

I. Introduction

IP multicast (IPMC) reduces the traffic load of one-to-many traffic [1], e.g., Multicast VPN, streaming, content delivery networks, or data center virtualization/overlay because it avoids redundant packet copies. To that end, it distributes traffic of a multicast group along a tree so that any link in an IP network forwards at most a single copy of a packet. However, all core nodes that are part of a distribution tree of an IPMC group need to maintain forwarding state for that IPMC group. This approach causes a threefold scalability issue. First, core nodes need to maintain possibly extensive forwarding information bases (FIBs). Second, when subscriber change, the core nodes of the affected IPMC group require updates which cause serious signaling efforts. Third, if links or nodes fail, or the topology changes, the traffic of many multicast groups may be affected so that many routers experience a large signaling load. The IETF has proposed Bit Index Explicit Replication (BIER) [2] to counteract that problem. BIER tunnels multicast traffic through a BIER domain and delivers a copy to each desired egress node. BIER solves the scalability problem by keeping the core nodes of the BIER domain unaware of any multicast group. Nevertheless, scaling BIER to large networks is a challenge. Multiple copies of a multicast packet may need to be forwarded over the same link, which contracts the multicast idea and may prevent BIER from efficiently reducing the traffic load for multicast traffic. We briefly explain the reason and provide the ground for this research work.

When an ingress node of a BIER domain receives an IPMC packet, it adds a BIER header including a bitstring. The positions in the bitstring correspond to egress nodes of the BIER domain and the activated bits indicate the receivers of the BIER packet. The bitstring enables BIER routers to forward BIER packets without knowing multicast groups. As the bitstring has a limited size, BIER domains with more egress nodes require a scaling feature. Subdomains are introduced which are sets of egress nodes, and bitstrings are defined for each subdomain. Thus, if an IPMC packet needs to be forwarded to egress nodes in different subdomains, multiple BIER packets with different bitstrings are sent and possibly pass identical links. This obviously reduces the efficiency of BIER to distribute multicast traffic compared to IPMC. Thus, BIER enables stateless transport of multicast traffic and thereby mitigates IPMC's scalability problem. However, it is less efficient than IPMC with regard to traffic load reduction.

The contributions of this paper are manifold. First, we show that a simple application of BIER's scaling feature [2], i.e., random subdomain clustering, cannot efficiently reduce traffic load in the network . Second, we present means to compute efficient subdomain clusterings. To that end, we describe an integer-linear program (ILP) that computes subdomain clustering in a way that minimizes the overall traffic load in the network. We also design a heuristic to approximate the solution of the ILP because it works only on small topologies. Third, we quantify and compare the ability of IPMC and BIER to efficiently reduce the load from multicast traffic in comparison to unicast. In particular, we evaluate the efficiency of BIER with the proposed subdomain clustering mechanisms and compare it to a naive application of BIER's scaling feature. We define suitable metrics and show that the efficiency of multicast depends on network topology and size as well as the size of the multicast groups. Fourth, we investigate the effect of link failures on the efficiency of BIER with optimized subdomains. This is interesting as link failures change the routing based on which the subdomains were optimized.

The remainder of the paper is structured as follows. In the next section we review related work. Section III gives a primer

The authors acknowledge the funding by the Deutsche Forschungsgemeinschaft (DFG) under grant ME2727/1-2. The authors alone are responsible for the content of the paper.

*These authors contributed equally

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

on BIER and shows that BIER generates a separate packet copy for almost every subdomain even for small multicast groups. In Section IV we propose algorithms to compute subdomains for BIER networks. We compare the algorithms with regard to runtime and quality in Section V. Section VI evaluates and compares the traffic savings potential of IPMC and BIER for multicast traffic. In Section VII we evaluate the efficiency of BIER in case of single link failures. Finally, we conclude the paper in Section VIII.

II. Related Work

We review advances for IPMC and BIER-based multicast and mention well-known clustering algorithms.

A. Advances for IPMC

Islam et al. [3] and Al-Saeed et al. [4] provide comprehensive surveys for multicast. Most of the cited papers discuss shortcomings of IPMC as already mentioned in the introduction, i.e., limited scalability in terms of signaling and state overhead. Many approaches aim to make traditional IPMC forwarding more efficient. Intelligent mechanisms for multicast tree building are presented to reduce the size of the forwarding information base (FIB), or efficient signaling mechanisms are proposed. However, they counteract the shortcomings of traditional IPMC only up to the point where the inherent design flaw of traditional IPMC, i.e., maintaining IPMC-group-dependent state in core devices, causes significant overhead, and therefore scalability issues.

Elmo [5] improves the scalability of traditional IPMC in data centers. Multicast group information is encoded in packet headers to reduce the FIB of core nodes by leveraging characteristic properties of data center topologies. The Avalanche Routing Algorithm (AvRA) [6] also leverages properties of data center networks to optimize link utilization of distribution trees. Dual-Structure Multicast (DuSM) [7] builds specialized forwarding structures for high-bandwidth and low-bandwidth flows. It improves scalability and link utilization in data centers.

Zhang et al. [8] optimize application layer multicast (ALM). They continuously monitor the application-specific distribution tree and update its structure according to the optimization objective of the multicast group. The authors of [9] study the distribution of delay-sensitive data with minimum latency. They propose a set of algorithms that construct minimum-delay trees for different kinds of application requirements like min-average, min-maximum, real-time requirements, etc. Li et al. [10] leverage the structure of data center networks to improve the scalability of traditional multicast. They optimize the forwarding tables by partitioning the multicast address space and aggregating multicast addresses at bottleneck switches. Kaafar et al. [11] present a new overlay multicast tree construction scheme. It leverages location-information of subscribers to build efficient distribution trees.

Software-Defined Multicast (SDM) [12] is a well-managed multicast platform. It is specialized on P2P-based video streaming for over-the-top and overlay-based live streaming services. In [13] traffic engineering features are added to SDM. Lin et al. [14] propose to share distribution trees between multicast groups to reduce the size of the FIB in core nodes and implement it in OpenFlow. Similarly, the authors of [15] leverage bloom filters to reduce the number of TCAM-entries in software-defined networks. Adaptive SDN-based SVC multicast (ASCast) [16] optimizes multicast forwarding for video live streaming by minimizing latency and delay. To that end, the authors propose an integer linear program for optimal tree building, and TCAM-based forwarding tables for fast packet processing. Humernbrum et al. [17] reduce the size of the FIB in some core nodes by introducing address translation from multicast addresses to unicast addresses at the last multicast hop. Jia et al. [18] reduce the size of the FIB in core nodes and facilitate efficient implementations. They leverage prime numbers and the Chinese remainder theorem to efficiently organize FIB structures. Steiner trees [19] are well-researched structures to build efficient multicast trees. Many papers modify and extend Steiner trees to build specialized multicast trees that minimize specific aspects like link costs [20], number of branch nodes [21], number of hops [22], delay [23], optimal placement of IPMC sources [24], or retransmission efficiency [25].

B. Advances for BIER

BIER uses a novel header and its forwarding behavior distinguishes substantially from IP forwarding. That is, BIER does not require per-IPMC-group-state in its core devices. Therefore, it does not suffer the same scalability issues as IPMC. Giorgetti et al. [26], [27] show a first implementation of BIER in OpenFlow. Merling et al. [28] present a BIER prototype for a P4-programmable software switch with a throughput of around 900 Mb/s. In a follow-up work [29] they implement BIER for the P4-programmable switching ASIC Tofino that supports 100 Gb/s throughput per port. They also propose how BIER traffic should be rerouted in case of failures, which has been adopted as IETF working group document [30].

The authors of [31] evaluate the retransmission efficiency of BIER when subscribers signal missing packets by negative acknowledgments, i.e., NACKs. Traditional IPMC leverages either unicast packets or retransmission to the entire multicast group when some subscribers signal NACKs. The BIER header allows to retransmit packets to specific subscribers only, i.e., NACK senders, while sending only one packet copy over each link. The authors find that BIER causes less overhead in terms of number of retransmitted packets and that it achieves better link utilization. Desmouceaux et al. [32] increase efficiency of retransmission with BIER by allowing intermediate nodes to resend packets, if possible, instead of resending the packet at the source. This significantly reduces the overall retransmission traffic.

Eckert et al. [33] propose tree engineering for BIER, i.e., BIER-TE. It leverages the BIER header to also encode the distribution tree of a packet in terms of traversed links. In [34] 1+1 protection for BIER is presented using maximally

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

redundant trees (MRTs). Traffic is distributed simultaneously over two disjoint trees so that packets are delivered even if one tree is compromised by a failure.

C. Clustering Algorithms

In this work we cluster receivers of BIER domains into subdomains. Karypis et. al. [35] present an algorithm to compute a bisection of a graph by performing a breadth-first search starting from two center nodes. The authors of [36] propose a similar method to compute k-partitions for arbitrary k, using k center nodes. Instead of two breadth-first searches, this algorithm performs k breadth-first searches in parallel. The resulting partitions tend to reduce the number of border nodes instead of cross-edges, which is a good property for load balancing. The approach is closely related to k-means clustering with Lloyd's algorithm [37]. The algorithm selects k center nodes and adds all nodes to the cluster with the nearest center node. The center nodes are readjusted to reflect the center of the clusters and this step is repeated until no changes occur. k-means clustering is not suitable for our problem, as cluster sizes cannot be limited. In contrast to that, the bubble-growing approach of [36] produces equal size partitions. The heuristic algorithm for BIER clustering in this work follows a similar approach.

III. Bit Index Explicit Replication (BIER)

In this section we introduce fundamentals of BIER and explain its scaling mechanism for large networks. In addition, we show that the mechanism tends to produce multiple BIER packets for a single IPMC packet, even for small multicast groups.

A. Overview

BIER is a domain-based mechanism to transport IPMC traffic over a so-called routing underlay network, e.g., an IP network [2]. Figure 1 shows the layered BIER architecture.

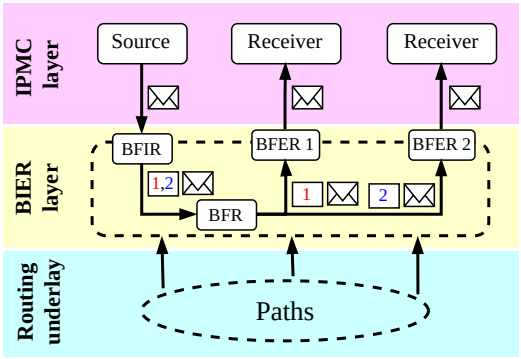


Fig. 1: Layered BIER architecture according to [28].

BIER-capable routers are called bit forwarding routers (BFRs). Ingress and egress nodes of a BIER domain are called bit forwarding ingress and egress routers (BFIRs, BFERs). The BIER header contains a bitstring with bit positions for all BFERs. BFIRs encapsulate IPMC traffic with a BIER header and the activated bits in its bitstring indicate the set of BFERs that are connected to subscribed IPMC clients, and hence, should receive a copy of the packet. BFRs forward BIER packets based on this bitstring along a tree towards the indicated BFERs. Thereby, only a single copy is sent over each involved link. The paths of the tree are inherited from the routing underlay but BIER-encapsulated IPMC packets are usually sent over Layer 2 technology. BFERs remove the BIER header from the packets and pass them to the IPMC layer.

B. Scaling BIER to Large Networks

BIER hardware must implement a bitstring length of 256 bits, but larger bitstrings, e.g., 1024 bits, may also be supported [2]. However, large bitstrings increase the header size, which is tolerable only to some extent. Any BFER requires a position in the bitstring to be addressable. To make BIER applicable to networks with more BFERs than the size of the bitstring, so-called BIER subdomains are introduced. BIER subdomains are identified by their subdomain identifier (SDI) and they define different mappings of BFERs to bit positions for the subdomain-specific bitstring in the BIER packet header. Therefore, only the combination of SDI and bitstring in the BIER packet header determines the addressed BFERs of that packet. If a BFIR receives an IPMC packet, it sends a packet copy of that IPMC packet to each subdomain that contains at least one receiver, i.e., BFER. Thereby, the BFIR encapsulates the packet copies with a BIER header with the right SDI and bitstring to address the subscribers in each subdomain.

C. BIER Packets Needed for Single IPMC Packet

When a BIER domain is large, it may require multiple subdomains. Then, the BFERs of a BIER domain are assigned to bit positions in the bitstrings of different subdomains. As a consequence, when an IPMC packet is to be carried through a BIER domain, multiple BIER packets with different SDIs may be created to address all desired receivers. We call them redundant packet copies as they carry the same IPMC packet. They cause extra traffic and reduce BIER's ability to reduce load from multicast traffic compared to normal IPMC forwarding.

We investigate how many different BIER packets are generated on average when a BFIR sends an IPMC packet over a BIER domain. To that end, we consider a BIER domain with $n = 1024$ BFERs and bitstring lengths of $b \in \{128, 256, 512, 1024\}$ bits. Hence, $s \in \{1, 2, 4, 8\}$ subdomains are needed to provide all BFERs with bit positions. We use a Markov chain model to compute the average number of different BIER packets needed if an IPMC packet has r BFERs as receivers; thereby we assume that receivers of a packet belong with equal probability to any of the subdomains.

Figure 2 shows that the average number of BIER packets significantly depends on the number of receivers r and the number of subdomains s. The number of BIER packets converges quickly to the number of subdomains s. If $r = 3 \cdot s$ receivers are addressed, almost s different BIER packets need to be sent for a single IPMC packet.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

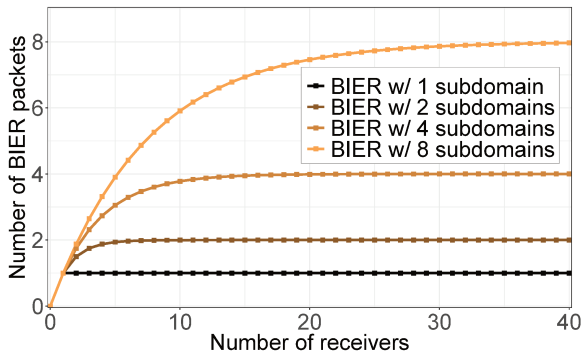


Fig. 2: Average number of redundant BIER packet copies needed to forward a single IPMC packet through a BIER domain with $n = 1024$ BFERs partitioned into $s \in \{1, 2, 4, 8\}$ subdomains.

As the number of redundant packets is large even for small multicast groups, it is relevant to study their impact on the overall extra traffic in the network and on the ability of BIER to efficiently carry multicast traffic compared to IPMC. Moreover, the effect of redundant BIER packets may be mitigated. If a specific part of the BIER domain accommodates only BFERs from a single subdomain, only packets for that subdomain will be forwarded to that part of the network, which avoids redundant packets in this area. Thus, when subdomains are chosen appropriately, BIER may be able to deliver multicast traffic with only little extra traffic compared to IPMC.

IV. Algorithms for BIER Clustering

As explained in the previous section, sudomains for BIER domains should be defined such that the overall load from multicast traffic is low in the entire network even if multiple redundant BIER packets need to be sent to BFERs in different subdomains.

We first formalize this challenge as the "BIER clustering problem". Then, we propose three classes of algorithms to assign BFERs to subdomains of a BIER domain: random, optimal, and heuristic. For optimal solutions, we propose topology-specific algorithms for selected, regular topologies. For arbitrary topologies we propose an integer linear program (ILP) to optimally solve the BIER clustering problem. Finally, we suggest a heuristic algorithm that may be used when the ILP is not solvable for complexity reasons.

A. The BIER Clustering Problem

We introduce nomenclature and constraints as well as the objective function for BIER clustering, and discuss alternate optimization goals. Finally, we discuss the investigated topology-types.

1) Nomenclature and Constraints: A network topology is given by a set of n vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. The set of edges on the path between any two nodes $v, w \in \mathcal{V}$ is denoted by $p(v, w) \subseteq \mathcal{E}$; it is inherited by the routing underlay. The objective of the clustering is to find a set of subdomains $\mathcal{C}$ so that any subdomain $\mathcal{S} \in \mathcal{C}$ is a subset of all nodes $\mathcal{S} \subseteq \mathcal{V}$ and the union of all subsets covers all nodes, i.e., $\bigcup_{\mathcal{S}\in\mathcal{C}} \mathcal{S} = \mathcal{V}$. Moreover, the size of a subdomain is limited by the length of the bitsring b.

In theory, there is no limit on the number of subdomains and subdomains may overlap. However, more subdomains imply more forwarding information on the BFRs, more complex bitstring definition for a multicast group, and longer subdomain identifiers. Therefore, we keep the number of subdomains as low as possible. We further assume that any node of the BIER domain is a BFER. Therefore, the number of subdomains is $s = \lceil \frac{n}{b} \rceil$. We require the subdomains to be disjoint. This simplifies the algorithms and has no impact on the results as the network sizes in our experiments are multiples of maximum subdomain sizes.

2) Objective Function: The objective function for the clustering is to keep the overall traffic low. We define the overall traffic as the number of packets carried over all links of the BIER domain when every node sends a packet to every other node. The traffic induced by a BIER packet sent from a single BFIR v to all BFERs within a subdomain $\mathcal{S}$ is the number of edges traversed, i.e., $\left|\bigcup_{w\in\mathcal{S}} p(v, w)\right|$. Thus, the overall BIER traffic load is

$$\rho = \sum_{v\in\mathcal{V}} \sum_{\mathcal{S}\in\mathcal{C}} \left| \bigcup_{w\in\mathcal{S}} p(v, w). \right| \tag{1}$$

This metric is to be minimized by a clustering $\mathcal{C}$.

3) Alternate Optimization Goals: Future work may optimize the clustering so that more subdomains are allowed. Then, some subdomains may overlap and BFIRs can choose over which subdomains a BFER will be reached. This increases the potential for the minimization of overall traffic so that even fewer redundant BIER packets may need to be generated. This, however, requires the knowledge of the IPMC groups and needs more resources on BFRs and BFIRs.

Another optimization approach is monitoring BIER traffic and building subdomains according to higher-level information, e.g., multicast groups, traffic patterns, proximity, etc. However, such methods are significantly more complex than the suggested approach, require additional research, and are out of scope of this paper.

4) Studied Topologies: We include two types of topologies in our study. First, we leverage full mesh, line, ring, and perfect binary tree to reveal interesting corner cases for multicast traffic. That is, multicast cannot reduce traffic in full-mesh topologies but has its greatest effect in line and ring topologies. Perfect-binary trees have a simple but non-random and non-trivial topology structure which is why we included them in the evaluations. The evaluations of such best/worst cases gives insights in upper and lower bounds. Second, we conduct experiments on sets of random mesh topologies with different average node degrees to analyze the performance of the studied mechanisms on more realistic topologies.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

B. Random BIER Clustering

We briefly explain random BIER clustering. A bitstring length of b is given. A set of n BFERs is subdivided into equal-size $s = \lceil \frac{n}{b} \rceil$ subdomains. BFERs are randomly assigned to these subdomains whereby their size is limited to b BFERs. In Section V-C we use this algorithm as a baseline for comparison.

C. Optimal BIER Clustering for Selected Topologies

We describe optimal clusterings for selected, regular topologies: full mesh, line, ring, and perfect binary tree. We renounce on a formal proof of optimality as this is rather obvious.

1) Full Mesh: Here, random assignment is optimal. In full meshes, all traffic is exchanged over a direct link between source and destination because all nodes are neighbors. However, in such topologies, there is no traffic reduction potential for multicast and we do not consider full meshes any further.

2) Line Topologies: Start at one end of the line. Assign the next b neighboring nodes to a subdomain. Repeat until all nodes are assigned. The last subdomain may have less than b nodes.

3) Ring Topologies: Select an arbitrary position in the ring and choose a direction. Assign the next b neighboring nodes to a subdomain. Repeat until all nodes are assigned. The last subdomain may have less than b nodes.

4) Perfect Binary Trees: We consider a perfect binary tree. The depth of a node is its distance to the root plus one so that the leaves have maximum depth. We denote their depth as the height h of the tree. We state that a perfect binary tree with height h has $2^h - 1$ nodes.

We assume that the bitstring size is $b = 2^k$. It can accommodate a perfect binary tree with height k. We give an algorithm to cluster a perfect binary tree with height h into 2^{h-k} subdomains with up to 2^k nodes. We take all subtrees with roots of depth $h-k+1$ as initial subdomains. The other unassigned nodes are assigned to a nearest possible subdomain which still accepts additional nodes. Thereby, the assignment order of these nodes is inverse to their depth. The order among nodes with equal depth does not matter.

D. Optimal BIER Clustering for Arbitrary Topologies

We first explain fundamentals of integer linear programs (ILPs). Then, we apply them for optimal clustering of BIER domains.

1) Fundamentals of ILPs: An ILP describes the solution space of an optimization problem with so-called decision variables and linear inequalities. Parameters of the optimization problem serve as coefficients in the inequalities. A linear objective function describes the quality of possible solutions and is to be minimized.

ILP solvers find the best integer solution for decision variables that fulfill all inequalities. During the solution process, an ILP solver indicates lower and upper bounds regarding the objective value for the best solution. The upper bound is the value for the best solution found so far. While progressing, better solutions may be found and the lower bound for the best solution may increase. If upper and lower bound meet, the ILP solver found an optimal solution.

2) BIER Clustering Using ILPs: We build an ILP that describes the solution space for BIER clustering and an objective function for the overall traffic load given in Equation (1). Its output is an optimal clustering $\mathcal{C}$ of the network that minimizes the objective function.

$$\forall v \in \mathcal{V}: \sum_{\mathcal{S} \in \mathcal{C}} x_v^{\mathcal{S}} = 1 \tag{2}$$

$$\mathcal{S} \in \mathcal{C}: \sum_{v \in \mathcal{V}} x_v^{\mathcal{S}} \leq b \tag{3}$$

$$\forall v, w \in \mathcal{V}, e \in \mathcal{E}, \mathcal{S} \in \mathcal{C}: \; p_{e,v,w} \cdot x_w^{\mathcal{S}} \leq y_{v,e}^{\mathcal{S}} \tag{4}$$

$$\forall v \in \mathcal{V}, e \in \mathcal{E}, \mathcal{S} \in \mathcal{C}: \; y_{v,e}^{\mathcal{S}} \leq \sum_{w \in \mathcal{V}} p_{e,v,w} \cdot x_w^{\mathcal{S}} \tag{5}$$

$$\text{min: } \rho = \sum_{v \in \mathcal{V}} \sum_{\mathcal{S} \in \mathcal{C}} \sum_{e \in \mathcal{E}} y_{v,e}^{\mathcal{S}} \tag{6}$$

The ILP is given by Equation (2), Inequalities (3)–(5), and the objective function (6). It contains two types of binary decision variables. The decision variable $x_v^{\mathcal{S}}$ indicates whether node v belongs to subdomain $\mathcal{S}$; it is 1 if $v \in \mathcal{V}$ is in subdomain $\mathcal{S}$, otherwise it is 0. Equation 2 enforces that any node is part of exactly one subdomain. Inequalitiy 3 ensures that a subdomain contains at most b nodes. The decision variable $y_{v,e}^{\mathcal{S}}$ indicates whether edge e is part of the multicast tree from node v to any node $w \in \mathcal{S}$. It depends on $x_v^{\mathcal{S}}$ and the forwarding information. The latter is given by coefficients $p_{e,v,w}$ which are 1 if edge e is on the path from v to w; otherwise the coefficient is 0. This dependency is modelled by Inequalities 4 and 5. Equation 4 ensures that $y_{v,e}^{\mathcal{S}} = 1$ if e is part of the path from BFIR v to any BFER w in subdomain $\mathcal{S}$. Equation 5 ensures that the decision variable $y_{v,e}^{\mathcal{S}}$ is 0 if e is not part of any path from v (BFIR) to any w (BFER) in $\mathcal{S}$; thereby the membership $w \in \mathcal{S}$ is expressed only indirectly by $w \in \mathcal{V}$ and the decision variable $x_w^{\mathcal{S}}$.

The objective function in Equation (6) quantifies the overall traffic as defined in Equation (1) and is to be minimized.

E. Heuristic BIER Clustering

We propose a heuristic clustering algorithm that consists of two phases. Phase 1 selects initial subdomains. Phase 2 improves these subdomains according to Equation (1) by exchanging the assignment of node pairs to their subdomains.

Phase 1 works as follows. First, randomly select s nodes as center nodes of the different subdomains. Second, add further nodes to the subdomains until their maximum size b is reached. To that end, nearest non-assigned nodes are assigned to the center nodes in round-robin fashion. This yields a clustering of the BIER domain into subdomains. We repeat Phase 1 to generate $10 \cdot s$[†] clusterings and choose the best according to

[†]We performed evaluations with significantly higher repetitions but observed no increase in quality. Thus, we selected 10 runs as a reasonable basis to find a good solution.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

the objective function in Equation (1) to continue with it in Phase 2.

Phase 2 improves the clustering. First, randomly select two nodes that have neighbors in other subdomains and that are assigned to different subdomains. Swap their assignment if this reduces the overall load according to Equation (1). Repeat this procedure until ρ from Equation (1) does not decrease for $n = |\mathcal{V}|$ steps. When computing a clustering for a network, we perform the presented algorithm 20 times and take the best solution.

This algorithm is simple but works better than more complex approaches we have evaluated before. We evaluate the quality of this heuristic in the next section.

V. Comparison of BIER Clustering Algorithms

In this section we compare the BIER clustering algorithms from the previous section with regard to runtime and quality. First we present the topologies that we use for evaluations in this paper. Then, we demonstrate that the runtime of the ILP-based optimization is feasible only for small networks. Finally, we compare the quality of the subdomains obtained for different algorithms, topologies, and network sizes.

A. Topologies

In this work we investigate delivery of multicast traffic in various network topologies: full mesh, line, ring, perfect binary tree, and mesh networks with node degree $d \in \{2, 4, 6, 8\}$. We refer to the latter as mesh-d. We construct them using the topology generator BRITE [38] which leverages a Waxman model [39]. While the first mentioned topologies are regular so that there is only a single choice for a network with n nodes, mesh-d networks are randomly constructed. Therefore, we generate 10 different representatives and compute average values for the considered metrics. The 95% confidence intervals are below 0.3% for all reported results so that we omit them in all tables and figures.

Topology	$n = 64$		$n = 128$	
	$s = 2$	$s = 4$	$s = 2$	$s = 4$
Line	0.11	3.80	1.07	45.51
Ring	66.51	21139.70	3633.59	-
Perfect binary tree	0.11	1.10	0.33	6.71
Mesh-2	0.06	3.59	0.21	22.67
Mesh-4	76.09	-	-	-
Mesh-6	718.23	-	-	-
Mesh-8	3883.62	-	-	-

Tab. 1: Time to solve ILPs for BIER clustering in seconds. Some instances could not be solved within 72 hours.

B. Runtime for ILP-Based Optimization

We measure the runtime to solve ILPs for BIER clustering with the ILP solver Gurobi 9.1 on a Ryzen 3900X CPU with 12 cores running at 3.8 GHz with 64 GB RAM.

Table 1 compiles the runtimes of the solver for different network topologies, network sizes, and number of subdomains. Perfect binary trees have one node less than indicated in the table. The runtime to solve the ILPs increases with network size and in particular with the number of subdomains. The network topology also has a significant impact. For some topologies, networks with 128 nodes or with 4 subdomains cannot be solved within three days.

In contrast, the heuristic algorithm has a runtime of a few seconds for any topology with $n = 1024$ nodes, and $s = 4$ subdomains. For the largest networks with $n = 8192$ nodes and $s = 32$ subdomains, it takes 8–16 h for mesh-4 and mesh-6, and 16–24 h for lines, perfect binary trees, mesh-2, and mesh-8. Only very large rings with $n = 8192$ nodes required around 3 days.

Thus, solving the ILP for optimal BIER clustering is infeasible for realistic problem instances, but the runtime of the heuristic algorithm is acceptable even for large networks. Therefore, we utilize for the evaluations in Section VI the topology-specific solutions of Section IV-C for lines, rings, and perfect binary trees, and the heuristic algorithm for mesh-d networks.

C. Quality Comparison

We now compare the quality of heuristic results with those from optimal and random subdomain assignment. The metric is the overall traffic load ρ with BIER when every node sends a packet to any other node (see Equation (1)).

We first consider mesh-d, for which only the ILP-based algorithm can deliver optimal results but only for small networks. Table 2 shows the overall traffic for subdomains generated with heuristic and with random assignment relative to the overall traffic for optimal subdomains. All heuristic results are close to optimum. We observe for mesh-2 that larger networks and more subdomains slightly degrade the results of the heuristic algorithm. Random assignment is clearly worse, i.e., it generates 33%-80% more extra traffic than optimal subdomains while heuristic assignment causes only 0.3%-1.5% more extra traffic. The quality of the heuristic results tends to improve with increasing node degree.

Topology	n	s	Heuristic (%)	Random (%)
Mesh-2	64	2	100.3	132.6
		4	100.7	162.2
	128	2	100.5	133.7
		4	101.5	179.8
Mesh-4	64	2	100.3	115.2
Mesh-6	64	2	100.4	110.6
Mesh-8	64	2	100.3	107.1

Tab. 2: Overall traffic load for heuristic and random BIER clustering depending on network size n and number of subdomains s; numbers are relative to the overall traffic load for optimal subdomains computed based on ILP solutions.

Now we discuss larger, regular topologies for which the algorithms of Section IV-C provide optimal results. We cluster the networks into subdomains of size $b = 256$. The results are compiled in Table 3. We consider networks with $n \in \{256, 512, 1024, 2048, 4096, 8192\}$ nodes, an exception are perfect binary trees with only $n - 1$ nodes. Consequently, multiple subdomains $s \in \{1, 2, 4, 8, 16, 32\}$ are needed. The

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

overall traffic load is given relative to the one for optimal subdomains.

n	s	Line (%)		Ring (%)		Perfect binary tree (%)	
		Heur.	Rnd.	Heur.	Rnd.	Heur.	Rnd.
256	1	100	100	100	100	100	100
512	2	100	159.5	100	133.1	101.2	142.8
1024	4	100	212.2	100	199.1	100.8	197.2
2048	8	100	249.3	100	265.1	100.6	262.5
4096	16	100	271.8	108.7	317.9	104.9	336.9
8192	32	100	284.3	134.1	353.0	118.0	416.9

Tab. 3: Overall traffic load for heuristic and random BIER clustering depending on network size n and number of subdomains s; numbers are relative to the overall traffic load for optimal subdomains computed based on topology-specific solutions.

We observe that the quality of the heuristic is almost optimal for up to 2048 nodes. Beyond that, the quality degrades by up to 34% for rings compared to optimum. The quality for lines and perfect binary trees is better with a degradation of at most 18%. The results with random assignment are much worse than those with optimum and heuristic assignment.

We draw two major conclusions. First, optimization of subdomains is important as random subdomains are likely to cause a lot more extra traffic than needed in large BIER domains. Second, subdomains obtained through the presented heuristic are almost optimal for networks up to 2048 nodes, beyond that we see a degradation. However, even then heuristic subdomains are still much better than random subdomains. The heuristic is needed for the evaluation of mesh-d networks in Section VI. We believe that the quality of the heuristic results for mesh-d is acceptable even for large networks because the heuristic algorithm performed well in large networks for lines, rings, and perfect binary trees. Therefore, the method may be suitable for application in practice.

VI. Traffic Savings with IPMC and BIER

In this section we investigate the potential of multicast variants, i.e., IPMC and BIER, to reduce the traffic load from multicast traffic relative to unicast, and compare it with each other. We first discuss the methodology. Afterwards we study the reduction potential for overall traffic depending on network size and multicast group size. Then, we show that both IPMC and BIER can well avoid heavily loaded links. Finally, we examine the impact of header size on the traffic saving potential of BIER.

A. Methodology

We describe the general evaluation approach, investigated network topologies, the way BIER subdomains are clustered in the study, packet sizes, evaluation metrics, and identified influencing factors.

1) General Approach: It is obvious that multicast groups can be very different, both in size and geographical distribution. Moreover, networks supporting multicast can have different topology. As those factors likely impact the efficiency of multicast variants, we study them depending on network topology, network size, and multicast group size. We study the topologies presented in Section V-A; if the topologies are random, we report averages from 10 different topologies and omit the small confidence intervals as mentioned. The networks have $n \in \{256, 512, 1024, 2048, 4096, 8192\}$ nodes, with the exception of perfect binary trees that have only network size $n - 1$.

In this section we evaluate the traffic saving potentials of IPMC and BIER in comparison to unicast and to each other. We simulate the transmission of a single packet from every source to all subscribers of a multicast group. We describe the models for multicast groups in the subsequent subsections as they depend on the experiments. The traffic handling is different for the three transport mechanisms unicast, traditional IPMC, and BIER. BIER is considered with subdomains which are explained in Section VI-A2. The impact of the three transport mechanisms is quantified by the overall network load and link loads. Both metrics are explained in Section VI-A3. Since load heavily depends on packet sizes we discuss them in Section VI-A4. Finally, we explain the investigated factors in Section VI-A5 which have an influence on the performance results.

2) BIER Clustering: We recap our findings from Sections IV and V, and describe how we configured BIER for the evaluations.

BIER without subdomains cannot support arbitrary topology sizes without extensive headers. BIER with subdomains supports large topologies but its efficiency heavily depends on the subdomain clustering (see Section IV-A). Therefore, we designed an integer-linear program (ILP) to find optimal subdomain clusterings (see Section IV-D) in arbitrary topologies and presented optimal BIER clusterings for selected topologies (see Section IV-C). However, the ILP can compute clusterings only in small networks due to runtime restrictions (see Section V-B). Therefore, we designed a heuristic for that purpose (see Section IV-E) and showed that its results are reasonably close to results from the ILP (see Section V-C).

For all following evaluations we consider only BIER forwarding with subdomains. On random topologies we compute the subdomain clustering with the proposed heuristic from Section IV-E. For selected topologies, i.e., ring, line and binary tree, we leverage the presented optimal clustering approaches from Section IV-C.

If not stated otherwise, we assume in our studies for BIER a bitstring size of $b = 256$ bits because that value must be supported by all BIER-capable equipment. Thus, b is also the maximum number of BFERs in subdomains. We assume that all nodes are BFERs. That means, when networks have more than b nodes, the nodes are partitioned into a minimum number of $s = \lceil \frac{n}{b} \rceil$ subdomains.

3) Metrics: We utilized two performance metrics in the simulations: overall traffic load and link load. We describe them in the following.

a) Overall Traffic Load: In Section VI-B we evaluate the overall traffic load. The overall load is the accumulated number of bytes sent in an experiment over any link in the network

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

to distribute the packets from each source to all receivers. This value obviously depends on the transport mechanism. To quantify the traffic savings potential of IPMC and BIER we relate their overall load for a specific traffic scenario to the one of unicast and to each other. For all evaluations, packets follow shortest path trees based on the hop count metric.

b) Link Load: Traffic load is not equally distributed over all links. Central links tend to have higher load than others so that they may profit more from traffic load reduction through multicast. Therefore, we study link load reduction on links in Section VI-C. To that end, we count packets carried over specific links instead of bytes as this facilitates interpretation of the results.

4) Packet Sizes: Table 4 shows the total packet sizes for different transport mechanisms in byte (B). For unicast and IPMC traffic we assume a packet size of 520 B which is the average size of IP packets on the Internet [40]. For BIER packets we assume a total size of 564 B, i.e., 520 B payload including the IPMC header plus 44 B to respect the additional BIER header fields and a bitstring length of 256 bits. If a longer bitstring length is used, the additional bytes are added to the 564 B.

Transport mechanism	IPMC packet w/ payload (B)	BIER-X (B)	Total packet size (B)
Unicast	520	-	520
IPMC	520	-	520
BIER-256	520	44	564
BIER-512	520	76	596
BIER-1024	520	140	660
BIER-2048	520	268	788
BIER-4096	520	524	1044
BIER-8192	520	1036	1556

Tab. 4: Total packet sizes for different transport mechanisms in byte (B). BIER-X refers to a BIER header with a bitstring of X bits. Thus, the total BIER header size is $(\mathrm{X}/8$ bits) plus 12 B for all other BIER header fields.

5) Investigated Factors: We investigate the following four factors. First, we consider different topology-types. That is, we selected line, ring and binary tree topologies to evaluate scenarios where distributing traffic with traditional IPMC or BIER has significant advantages due to the many shared paths of packets. Furthermore, we investigate random topologies with different average node degrees. This factor is relevant for all following evaluations.

Second, we evaluate all mechanisms on different network sizes to determine the scalability in large networks (see Section VI-B1). Third, we vary the size of multicast groups (see Section VI-B2), and thereby the number of receivers. That is, not every node in the network may necessarily be a subscriber which influences the efficiency of the transport mechanisms.

Finally, we investigate the impact of the BIER header size. On the one hand, small BIER headers add only little overhead, on the other hand large BIER headers reduce the number of extra copies needed to reach all subscribers in large networks. We study this tradeoff in Section VI-D.

We evaluate those factors by keeping other factors stable and varying the desired parameter. For example, we chose full multicast groups and change only the network size to determine its impact on the metric.

B. Reduction of Overall Traffic

We evaluate the potential for the reduction of overall traffic through multicast variants relative to unicast and compare the efficiency of BIER with the one of IPMC. To that end, we measure the number of transmitted bytes in the network to distribute a packet from all sources to all destinations (see Section VI-A3a). We first study the impact of network topology and size and then the impact of network topology and multicast group size.

1) Impact of Network Size: We evaluate the savings potential for overall traffic through multicast variants. To that end, we consider different network topologies and sizes and maximum multicast groups. That is, every node is a subscriber and receiver. We study IPMC vs. unicast, BIER vs. unicast, and BIER vs. IPMC.

a) IPMC vs. Unicast: Figure 3(a) shows the overall traffic for IPMC relative to unicast for multiple network topologies depending on the network size. The IPMC traffic load decreases relative to the unicast traffic load with increasing network size. There is a large reduction potential in line and ring networks so that the IPMC traffic volume is less than 2% compared to the one of unicast. In perfect binary trees the traffic can be reduced to 10% for $n = 255$ nodes and to 5% for $n = 8191$ nodes. Random mesh networks have a lower reduction potential that decreases with increasing node degree.

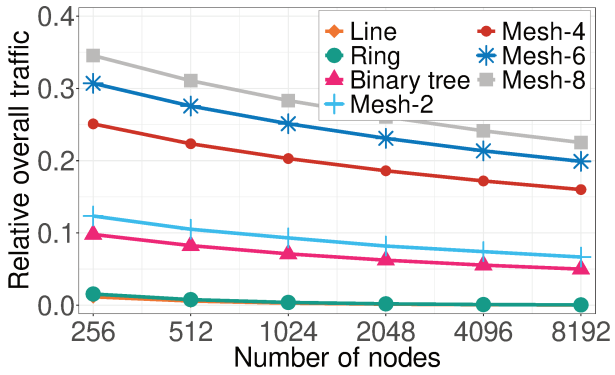
We observe an obvious dependence of the traffic reduction potential of IPMC on the network topology. We show that it is $\frac{1}{l}$ in the presence of maximum multicast groups. Multicast requires $n-1$ hops to distribute a packet from one source to $n-1$ receivers as this is the number of links in any shortest-path tree. Thus, $n \cdot (n-1)$ hops are required to distribute a packet from each node to all other nodes. When the same is done with unicast, any source node $v \in \mathcal{V}$ sends a packet to any destination node $w \in \mathcal{V}$. This requires $|p(v,w)|$ hops per v/w pair, which is in sum

$$\sum_{v\in\mathcal{V}}\sum_{w\in\mathcal{V}} |p(v,w)| = n\cdot(n-1)\cdot \frac{\sum_{v\in\mathcal{V}}\sum_{w\in\mathcal{V}} |p(v,w)|}{n\cdot(n-1)} = n\cdot(n-1)\cdot l. \quad (7)$$

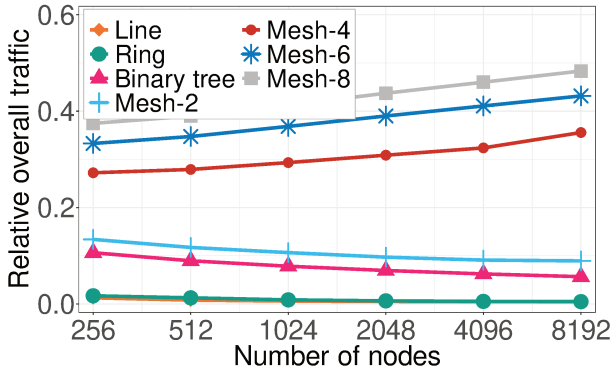
This follows that IPMC can reduce the overall traffic to $\frac{1}{l}$ compared to unicast. Lines and rings have by far the longest average path length and it strongly increases with increasing network size. In other topologies, average path lengths are clearly lower and increase slowly with the network size. The average path length correlates with the node degree and increases in the following topologies: mesh-8, mesh-6, mesh-4, mesh-2 and perfect binary trees.

b) BIER vs. Unicast: Figure 3(b) presents the overall traffic load for BIER relative to unicast. The number of required subdomains increases with the network size and thereby

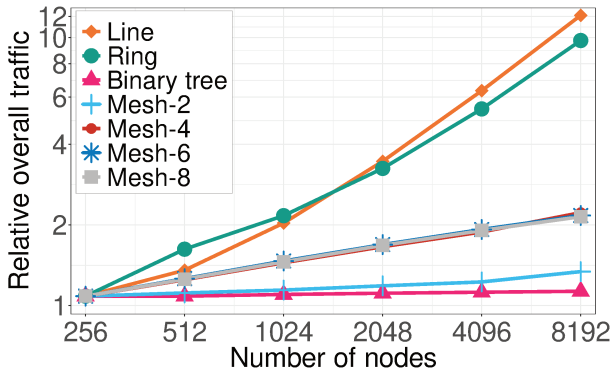
This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294



(a) IPMC relative to unicast. The curves for line and ring are almost identical.



(b) BIER relative to unicast. The curves for line and ring are almost identical.



(c) BIER relative to IPMC.

Fig. 3: Overall relative traffic load depending on the network size; every node sends one packet to all other nodes.

the number of BIER packets needed for these subdomains. We observe that BIER achieves the largest traffic reduction in lines and rings (around 97%), followed by perfect binary trees and mesh networks with node degree 2 (around 90%). Mesh networks with node degrees 4, 6, and 8 have a lower savings potential of around 50% and BIER's traffic savings potential relative to unicast decreases with increasing network size. The latter is different to IPMC (cf. Figure 3(a)). Thus, BIER can reduce the load of multicast traffic similarly well as IPMC, except in large, highly meshed networks in spite of well clustered subdomains.

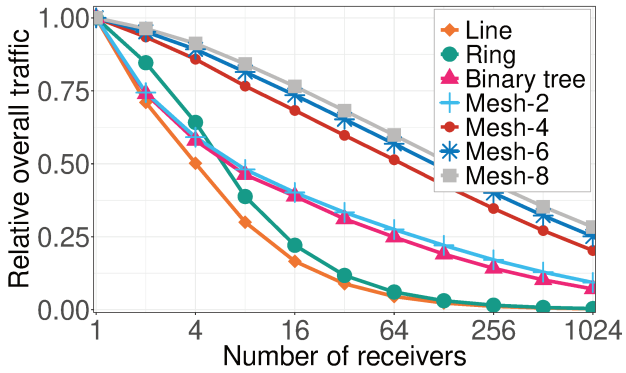
c) BIER vs. IPMC: To compare BIER directly with IPMC, we consider the fraction of overall traffic load of BIER and the one of IPMC in Figure 3(c). In line and ring networks, BIER causes a multiple (3 - 11 times) of the traffic that occurs with IPMC if the number of subdomains is very large, i.e., 8, 16, and 32 for network sizes of 2048, 4096, and 8192 nodes. That is because BIER with subdomains causes redundant packet copies and the BIER header adds extra bytes which have to be transmitted. However, the previous evaluation shows that the savings compared to unicast are still enormous, i.e., more than 98%. In mesh networks with node degee 4, 6, and 8, the traffic load with BIER compared to IPMC increases about logarithmically with network the network size and roughly doubles the load with IPMC in very large networks. In perfect binary trees and mesh networks with node degree 2, the traffic load with BIER relative to IPMC also increases with network size, but BIER causes only 40% more traffic than IPMC in very large networks although up to 32 subdomains are supported.

2) Impact of Multicast Group Size: We evaluate the influence of the multicast group size on the traffic reduction potential of multicast variants. We perform this study for different network topologies and a network size of $n = 1024$. Every node sends a packet to a random set of receivers. The set sizes are $r \in \{1, 3, 7, 15, 31, 63, 127, 255, 511, 1023\}$ large, for perfect binary trees the maximum number of receivers is $r = 1022$. As these are random experiments, we run each experiment 20 times to obtain very small confidence intervals that we omit in the figures for the sake of clarity.

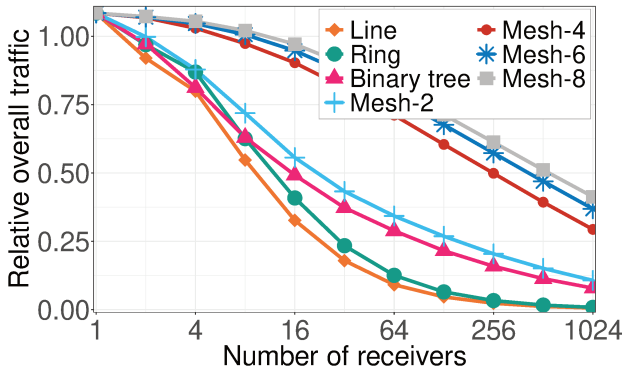
a) IPMC vs. Unicast: Figure 4(a) shows the overall traffic load of IPMC relative to unicast. It decreases with increasing multicast group size. For small multicast group sizes, IPMC can reduce the overall traffic only by little. In line and ring networks, large traffic savings are achieved already for small multicast groups, followed by perfect binary trees and random meshes with node degree 2. Random meshes with node degree 4, 6, and 8 require rather large multicast groups to provide a substantial savings potential.

b) BIER vs. Unicast: Figure 4(b) shows the overall traffic load for BIER vs. unicast. The figure looks similar to Figure 4(a), but all lines are slightly higher than with IPMC as the BIER header induces additional overhead. In particular, for a small number of receivers the overall traffic load with BIER relative to unicast is larger than 1 because the overhead generated by the BIER header is larger than the overhead saved

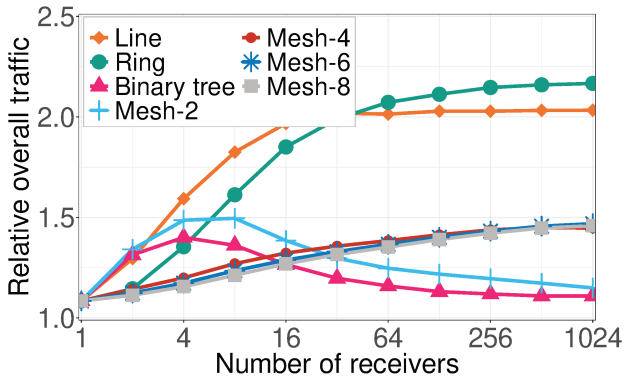
This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294



(a) IPMC relative to unicast.



(b) BIER relative to unicast.



(c) BIER relative to IPMC.

Fig. 4: Overall traffic load depending on the size of the multicast group; the networks are n = 1024 nodes large and every node sends one packet to the given number of random receivers.

through BIER in terms of saved packet copies.

c) BIER vs. IPMC: Figure 4(c) shows the overall traffic load with BIER compared to the one of IPMC. For a single receiver, BIER and IPMC are almost equally efficient as BIER does not send any redundant packets and the BIER header adds only a small overhead. . When the number of receivers increases, the overhead of BIER increases as more redundant BIER copies are sent. The values for $r = 1024$ receivers are same as the values for network size $n = 1024$ in Figure 3(c). Apart from that, BIER's overhead compared to IPMC depends on the network topology. For line and ring topologies, the overhead of BIER relative to IPMC increases up to a multicast group size of 64 receivers and remains constant afterwards. For mesh networks with node degree 4, 6, and 8, the overhead increases logarithmically with increasing number of subscribers. And for line and ring networks, the overhead decreases when the number of subscribers is 8 receivers or more.

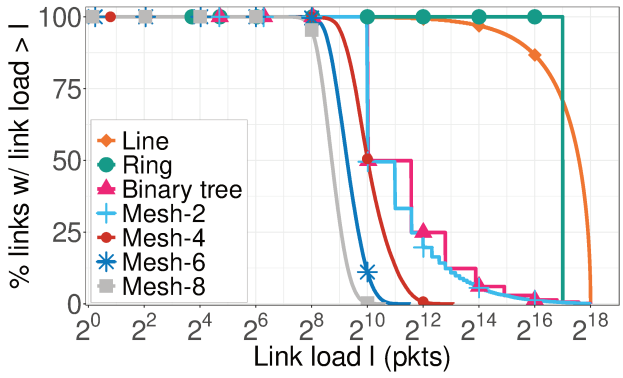
C. Avoidance of Heavily Loaded Links

For some network topologies, the savings potential through multicast is only moderate. However, traffic is not equally distributed over all links of a network as central links tend to carry more traffic. We show that multicast variants can greatly reduce the load on those links compared to unicast. We consider again maximum multicast groups and networks with $n = 1024$ nodes, $n = 1023$ for perfect binary trees, where every node is subscriber and receiver. We count the number of bytes carried over each link and discuss the complementary cumulative distribution function (CCDF) of the link loads (see Section VI-A3b) for unicast, IPMC, and BIER. In case of mesh-d, the load information of multiple networks is integrated in a single CCDF.

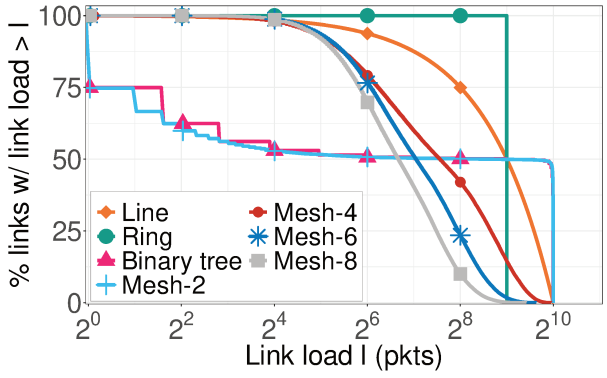
1) Link Load Distribution with Unicast: The CCDF for link loads with unicast is illustrated in Figure 5(a). In line and ring networks, a large percentage of links carries a large number of packets. With rings, any link carries the same number of packets due to symmetry. In perfect binary trees the number of packets per link is clearly lower than in lines and rings. Half of the links has only a load of 1022 packets, those are adjacent to the leaves. There are also a few links with a very high link load, those are links close the root. Random mesh networks with a node degree 2 have a similar CCDF as perfect binary trees. The random mesh networks with a node degree of 4, 6, and 8 have increasingly lower link loads and less variation regarding link loads.

2) Link Load Distribution with IPMC: Figure 5(b) shows the CCDF for link load with IPMC. The upper limit is now 1023 packets. Again, many links in lines and rings carry a large number of packets. In perfect binary trees and random mesh networks with node degree 2, 25% of the links have a very low load while 50% have a high load. In perfect binary trees, those are links close to the leaves and to the root, respectively. Random meshes with node degree 4, 6, and 8 reveal again a load continuum but it is at a lower load level compared to unicast. The x-scales in Figures 5(a) and 5(b) are

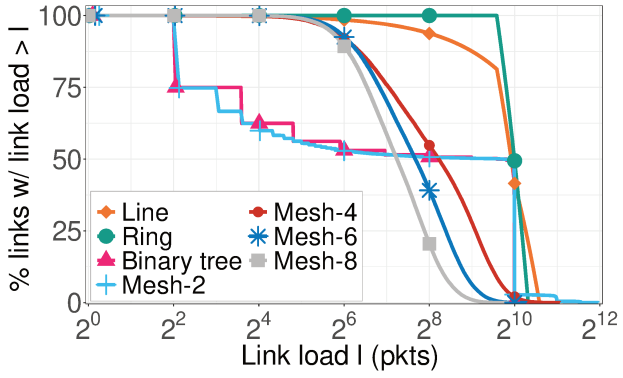
This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294



(a) Unicast.



(b) IPMC.



(c) BIER.

Fig. 5: Complementary cumulative distribution function (CCDF) for link loads in networks with 1024 nodes; every node sends one packet to all other nodes.

different. This suggests that there are many links for which IPMC decreases the traffic load by orders of magnitude. Thus, IPMC avoids in particular heavily loaded links compared to unicast.

3) Link Load Distribution with BIER: Figure 5(c) shows the CCDF of link loads with BIER. It looks similar to the one of IPMC in that the link load is mostly limited to 1023 packets. However, some links in line and ring networks have larger loads up to around 1600 packets, and a very few links in mesh networks with node degree 2 have loads of up to 3976 packets, i.e., roughly the maximum link load for multicast multiplied by the number of subdomains s. Mesh networks with a node degree of 4, 6, and 8 generally lead to lower link loads as their traffic is not concentrated on a few links.

The most important finding is that BIER also avoids very high loads on links compared to unicast. Only a very few links experience substantially higher link loads than with IPMC. Thus, BIER efficiently avoids heavily loaded links in a similar way as IPMC.

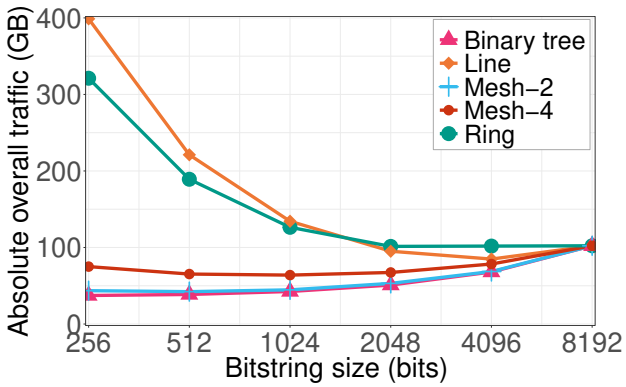
D. Impact of BIER Header Size

On the one hand, BIER largely avoids redundant packets over links, on the other hand BIER causes additional header overhead. There is an obvious tradeoff regarding header size: small bitstrings add only little header overhead, but require many redundant packets in large network, large bitstrings add lots of header overhead, but require only a few redundant packets in large networks. We expect an optimal header size in between. To study this effect, we first explain our methodology and then discuss experimental results.

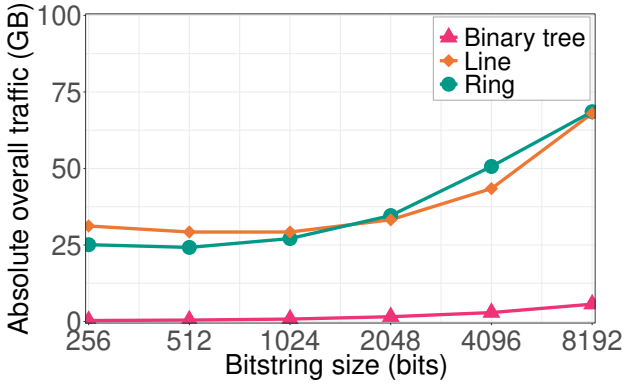
1) Methodology: We consider networks with $n = 8192$ nodes, $n = 8191$ for perfect binary trees. We investigate different bitstring sizes of $b \in \{256, 512, 1024, 2048, 4096, 8192\}$ bits that require $\lceil \frac{n}{b} \rceil$ SDs. We measure the total overall traffic in bytes as performance metric as this captures the effect of the BIER header size. We omit mesh-6 and mesh-8 topologies because results were almost the same as for mesh-4.

2) Results: We first study maximum-size multicast groups and each participant sends a single packet. Figure 6(a) shows the overall traffic volume for different bitstring sizes. The ring and the line topology lead to a large traffic volume for small bitstring sizes b. This results from long paths of most of the packets replicated for the $\lceil \frac{n}{b} \rceil$ SDs. Larger bitstring sizes reduce the number of SDs and thereby the replicated packets as well as the traffic volume. The optimal bitstring size for the line is $b = 4096$ bits and the one for the ring is $b = 2048$ bits. Larger bitstring sizes add so much header overhead that the overall traffic increases again. Topologies with shorter paths like binary trees, mesh-2 or mesh-4 networks reveal a clearly lower traffic volume for small bitstring sizes than lines or rings. The optimal bitstring size is $b = 256$ bits for binary trees, it is $b = 512$ bits for mesh-2, and it is $b = 1024$ bits for mesh-4. However, suboptimal bitstring sizes between 256 and 2048 bits lead only to slightly larger traffic volumes. Thus, any of these bitstring sizes is suitable for typical network topologies.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294



(a) Maximum-size multicast groups.



(b) Multicast groups with 128 random receivers.

Fig. 6: Overall traffic load depending on the bitstring size in the BIER header in networks with $n = 8192$ nodes.

The findings slightly change when we consider smaller multicast groups, i.e., multicast groups consisting of 128 randomly selected nodes. The corresponding results are shown in Figure 6(b). First, the overall traffic volume is clearly decreased as the number of receivers is lower (1.56%). Further, the optimum bitstring sizes are smaller, namely $b = 512$ for the ring and $b = 1024$ for the line. Thus, the optimum bitstring size depends on the size of the multicast groups. Therefore, the bitstring size is hard to optimize for practical applications when the size of the multicast groups is not known a priori. However, if the multicast groups are small, a small bitstring is recommendable. This makes the application of subdomains and their optimization even more relevant.

VII. Impact of Single Link Failures

We have optimized BIER subdomains for failure-free forwarding. In case of link failures, rerouting occurs in IP networks and then traffic is diverted around failed links. As a consequence, individual link loads and overall traffic load may increase. BIER with subdomains optimized for failure-free routing may lead to an even larger traffic increase than IPMC forwarding. Therefore, BIER may require more backup capacity than IPMC. We investigate these issues in the following. We first explain our methodology. Then, we perform simulations to study the overall traffic load and maximum link loads in case of single link failures, as well as the overall backup capacity required to accommodate rerouted traffic.

A. Methodology

Single link failures may partition a network topology. Then multicast groups are also partitioned into subgroups that cannot reach each other anymore. This can be avoided in resilient networks with 2-link-connected topologies and rerouting after failure detection. Thereby, end-to-end connectivity is not impaired so that participants of a multicast group can still reach each other. As a consequence, we consider only 2-link-connected topologies in this context, i.e., networks which are still connected after any single link failure. As a consequence, we do not consider lines and binary trees as they may be partitioned through single link failures. Rings are 2-link-connected by definition. We reuse the mesh-$\{4,6,8\}$ topologies from Section V-A which were chosen for the entire study such that they are 2-link-connected.

We consider networks with 1024 nodes and a bitstring with $b = 256$ bits. We optimize the subdomains for the failure-free case because it is the most common network state. That is, we use the heuristic clustering algorithms from Section IV-E for mesh-d topologies and the optimal clustering algorithm from Section IV-C for the ring topology. We compute subdomains based on the intact topology and evaluate BIER's efficiency when links in the network fail. We assume again a full multicast group and each participant sends a single packet to all other participants. We compute the effect of all single link failures for the mentioned topologies. That means, we remove the failed link from the topology, calculate new shortest paths, and compute the overall traffic load (see Section VII-B) and the maximum load increase on links (see Section VII-C); thereby, the subdomains remain unchanged. As mesh-d topologies are random, we report averaged results for them from 10 different topology samples.

In our experiments we count number of packets carried over links. When we extend the single sent packets to flows, we obtain observed rates which are proportional to the numbers of counted packets. To be more intuitive, we sometimes talk about rates and required capacities rather than counted packets, in particular when it comes to backup resources.

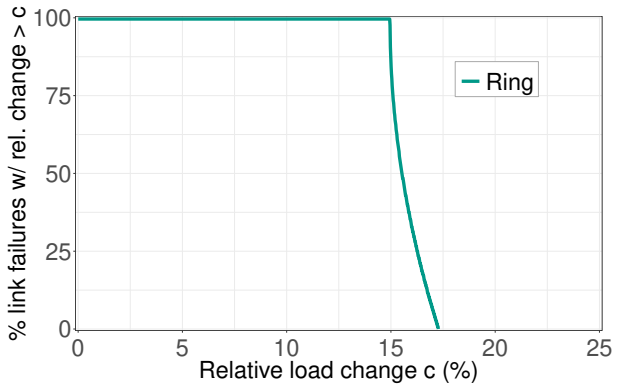
B. Overall Traffic Load

Traffic rerouting due to link failures possibly leads to longer paths, which may increase the overall link load in the network. Thus, we quantify the impact of single link failures on the overall traffic load (Equation 1) and compare it to the failure-free case both for IPMC and for BIER.

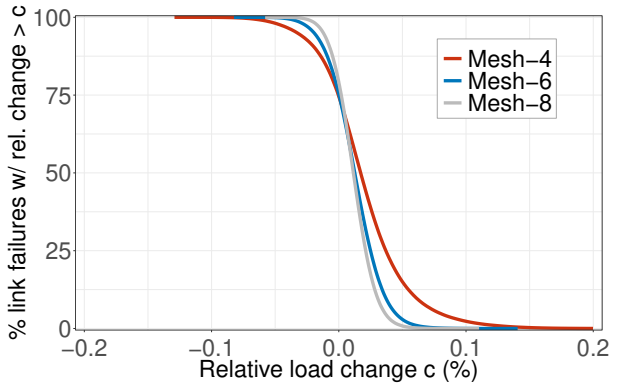
As the multicast groups in our experiments contain every node in the network, the overall traffic load for IPMC is $n \cdot (n-1)$ packets, no matter if a link fails. This is due to the fact that n packets are each forwarded along a single shortest path tree, and each shortest path tree consists of $n-1$ hops.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

Thus, the traffic load does not increase with IPMC in case of single link failures.



(a) Ring.

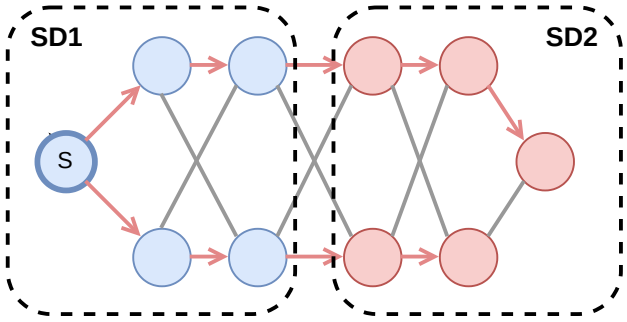


(b) Mesh-d.

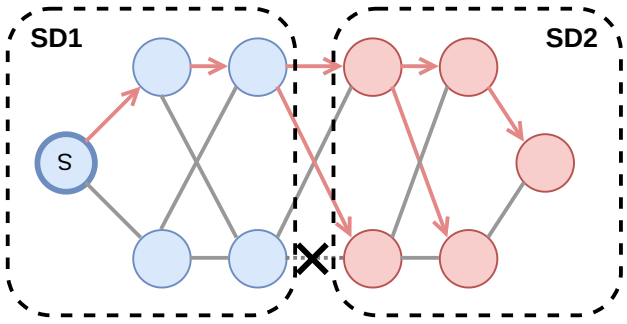
Fig. 7: BIER with single link failures – CCDFs of relative overall traffic change compared to the failure-free case, accumulated over all single link failures. Experiments are conducted in networks with $n = 1024$ nodes, every node sends a packet to every other node.

This is different with BIER. With BIER, $\lceil \frac{1024}{256} \rceil = 4$ packet copies, one for each subdomain, are forwarded over shortest path trees which consist of fewer hops than $n - 1$. However, their overall number of hops may change when traffic is rerouted. Therefore, we evaluate the change of overall traffic load with BIER for all single link failures. Figures 7(a) and 7(b) show CCDFs of relative overall traffic changes accumulated over all single link failures. We first discuss Figure 7(a) for a ring network. The overall traffic load rises between 15% and 17.3% depending on the position of the failed link. We explain this large increase as follows. Between any two nodes, there are exactly two paths in a ring network and the paths may have significantly different length. If the shorter path fails, traffic is rerouted over the longer path. This causes path stretch and leads to the observed increase in overall traffic load.

We now study mesh-d topologies for which the CCDF of the change in overall traffic load is presented in Figure 7(b). The increase in overall traffic load is bounded by 0.2%. We explain this as follows. In meshed networks with a node degree between 4 and 8, multiple paths exist between any two nodes and their lengths are likely to be similar. If the shortest path fails, another path with similar length is mostly available, which hardly increases the overall traffic load.



(a) Failure-free case.



(b) Single link failure.

Fig. 8: Example network with two BIER subdomains. In case of the indicated link failure, the adapted shortest path tree for the nodes in SD2 contains fewer hops than in the failure-free case, which reduces the traffic load.

We further observe that in 75% of all single link failures, the overall traffic load increases but in 25% the overall load decreases. This observation does not seem intuitive as the shortest path length for any pair of nodes remains unchanged or increases in case of a link failure. Nevertheless, the load may decrease as the shortest path tree towards the nodes in a subdomain may be more compact after rerouting. We illustrate this claim by the example in Figures 8(a) and 8(b). They show a network partitioned into two subdomains, SD1 and SD2. The shortest path tree starting in node S towards all nodes in SD2 contains two hops less in case of the considered link failure (Figure 8(b)) than under failure-free conditions (Figure 8(a)). This apparently more favorable path layout cannot be utilized under failure-free conditions because BIER traffic is always forwarded according to the paths in the underlay.

C. Maximum Load Increase on Links

When traffic is rerouted over another path, the traffic load on the corresponding links increases. We record for each link

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

the maximum load increase observed for any single link failure as this constitutes the required backup capacity for this link.

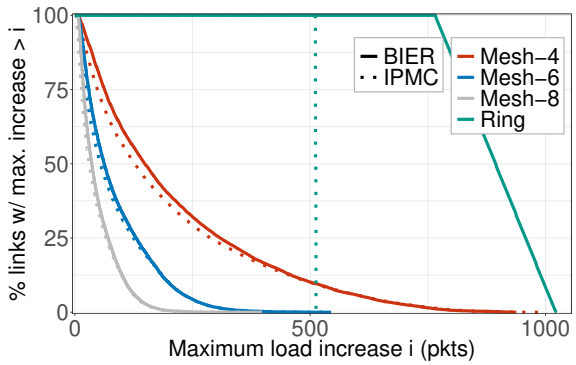


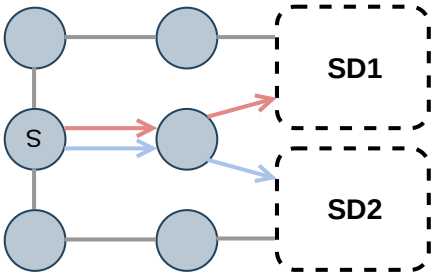
Fig. 9: CCDFs of maximum link load increases for single link failures. Experiments are conducted in networks with $n = 1024$ nodes, every node sends a packet to every other node.

Figure 9 shows the CCDFs of the maximum load increases for all links. In ring networks, all links experience up to 512 more packets with IPMC in case of link failures. In contrast with BIER, links carry between 768 and 1024 more packets. This is because multiple redundant BIER packets may be affected by the failure and are redirected. Therefore, BIER requires substantially more backup capacity in rings than IPMC and the exact amount depends on the location of a link within its subdomain. In mesh-d networks, the CCDF is almost a continuum. In networks with larger node degree, links require less backup than in networks with smaller node degree. This is due to shorter paths and less affected traffic, shorter backup paths, and better traffic distribution in case of link failures. Most notably, BIER causes about the same maximum load increases as IPMC although BIER requires more capacity than IPMC under failure-free conditions. We explain this fact by an example. Figure 10(a) shows a link carrying redundant BIER packets to two different subdomains. When that link fails, the traffic is redirected over different paths to the subdomains. IPMC would save a packet copy in the failure-free case, but it results into the same traffic distribution in this particular example.

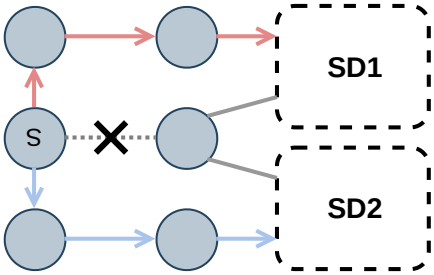
D. Overall Backup Capacity

We sum up link capacities for a network needed to carry the considered traffic for failure-free conditions on the one hand (capacity w/o backup) and for all single link failures on the other hand (capacity w/ backup). The difference is the absolute backup capacity. Table 5 compiles them for BIER and IPMC in mesh-d and ring topologies. The relative backup capacity is the ratio between absolute backup capacity and capacity w/o backup.

The results show that IPMC require 100% relative backup capacities for rings, but only 77%, 49%, and 36% for mesh-4, mesh-6, and mesh-8 networks. In contrast, BIER needs 176% backup capacity for rings, and 62%, 38%, and 29% for mesh-4, mesh-6, and mesh-8 networks. This is less than for IPMC,



(a) Failure-free case: two redundant packets are delivered over a link to two different subdomains.



(b) Single link failure: the two packets are redirected over different backup paths.

Fig. 10: Example network with two BIER subdomains. Redundant BIER packets for different subdomains are redirected over different paths.

	Metric	Ring	Mesh-4	Mesh-6	Mesh-8
IPMC	Cap. w/o backup	1047552	1047552	1047552	1047552
	Cap. w/ backup	2095104	1857633	1559138	1422539
	Abs. backup cap.	1047552	810081	511586	374987
	Rel. backup cap.	1.00	0.77	0.49	0.36
BIER	Cap. w/o backup	1051129	1395817	1418709	1406915
	Cap. w/ backup	2881534	2263645	1962557	1813694
	Abs. backup cap.	1830405	867828	543848	406779
	Rel. backup cap.	1.74	0.62	0.38	0.29
BIER / IPMC	Fraction w/o backup	1.003	1.33	1.35	1.34
	Fraction w/ backup	1.375	1.22	1.26	1.27

Tab. 5: Overall capacity w/ and w/o backup as well as absolute and relative backup capacity for IPMC and BIER. Capacities are given in packets.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

which seems surprising, but there is a simple explanation. BIER requires more capacity w/o backup than IPMC, but only little more backup capacity than IPMC. As a consequence, BIER's relative backup capacity is lower than the one for IPMC.

Below the line, BIER does not lead to excessive backup capacity demands when BIER subdomains are optimized for failure-free scenarios. The relative backup capacity is even lower than with IPMC. The ring network is an exception, but also IPMC requires lots of capacity in rings.

VIII. Conclusion

BIER is a novel forwarding paradigm to carry IP multicast (IPMC) traffic within so-called BIER domains. It is more scalable than IPMC because core nodes remain unaware of individual multicast groups. A problem arises for large BIER domains where subdomains need to be defined to make all egress nodes addressable. When an IPMC packet is distributed via a BIER domain, a separate BIER packet is needed for each subdomain that has a receiver for the IPMC packet. This leads to redundant packets and we showed that their number almost equals the number of subdomains if multicast groups are about 3 times larger than the number of subdomains. These redundant packets can significantly degrade BIER's ability to efficiently carry multicast traffic.

We argued that an appropriate choice of the subdomains can mitigate that effect when multiple BIER packets are sent to different regions of a network. Therefore, we defined the BIER clustering problem and proposed several algorithms to cluster a BIER domain into appropriate subdomains. We compared the runtime and quality of these algorithms, and showed that optimization of subdomains can greatly reduce the resulting overall traffic compared to random subdomains.

We evaluated and compared the ability of IPMC and BIER to reduce traffic load for multicast traffic relative to unicast transmission in different network topologies. It depends on the average path length in the network. IPMC can save lots of traffic in line and ring networks, in binary trees and in mesh networks with a low node degree. In mesh networks with larger node degree the traffic savings potential is smaller. It also depends on the network size. As BIER possibly sends redundant packets in large domains, its ability to reduce traffic load diminishes compared to IPMC. This also depends on network topology and size. In large networks with 8192 nodes and subdomain sizes of 256 nodes, BIER causes only moderate extra traffic compared to IPMC in binary trees and mesh networks with small node degrees. In contrast, it produces 10-12 times more traffic than IPMC in lines and rings, but the traffic savings potential of BIER is still very large in these topologies ($\sim$ 98%). In mesh networks with larger node degrees BIER doubles the overall traffic compared to IPMC and also the traffic savings potential is clearly reduced. These findings hold for maximum multicast groups. In smaller multicast groups the traffic savings potential of IPMC and BIER relative to unicast transmission is lower. While unicast causes enormous traffic loads on some links, both IPMC and BIER decrease such loads by orders of magnitude. The residual load on these links is higher with BIER than with IPMC due to redundant packets, but it is still on a low level.

We showed that there is an optimum size for the BIER header which depends on the network topology and on the size of the multicast groups. When multicast groups are rather small, small BIER headers are optimal, which makes the use of subdomains and their optimum selection more relevant.

We investigated the impact of single link failures on BIER domains with optimized subdomains. Rerouting causes only little more traffic load and the backup capacity needed for BIER networks is only little more tha the one of pure IPMC networks.

Below the line, subdomains are a good means to scale BIER to large networks, but they need to be carefully chosen to minimize extra traffic due to redundant packets.

Further studies may improve BIER clustering algorithms with regard to quality. They may also consider alternate optimization goals such as the ability to take advantage of overlapping subdomains for known multicast groups. Furthermore, scaling BIER-TE is a related problem but it requires different approaches.

References

[1] N. K. Nainar, R. Asati, M. Chen, X. Xu, A. Dolganow, T. Przygienda, A. Gulko, D. Robinson, V. Arya, and C. Bestler, *BIER Use Cases*, https://datatracker.ietf.org/doc/draft-ietf-bier-use-cases/12/, Sep. 2020.

[2] I. Wijnands *et al.*, *RFC 8279: Multicast Using Bit Index Explicit Replication (BIER)*, https://datatracker.ietf.org/doc/rfc8279/, Nov. 2017.

[3] S. Islam *et al.*, "A Survey on Multicasting in Software-Defined Networking," *IEEE Communications Surveys Tutorials (COMST)*, vol. 20, 2018.

[4] Z. Al-Saeed *et al.*, "Multicasting in Software Defined Networks: A Comprehensive Survey," *Journal of Network and Computer Applications (JNCA)*, vol. 104, 2018.

[5] M. Shahbaz *et al.*, "Elmo: Source Routed Multicast for Public Clouds," in *ACM SIGCOMM*, 2019.

[6] A. Iyer *et al.*, "Avalanche: Data Center Multicast using Software Defined Networking," in *International Conference on Communication Systems and Networks*, 2014.

[7] W. Cui *et al.*, "Scalable and Load-Balanced Data Center Multicast," in *IEEE GLOBECOM*, 2015.

[8] X. Zhang *et al.*, "A Centralized Optimization Solution for Application Layer Multicast Tree," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 14, 2017.

[9] K. Mokhtarian *et al.*, "Minimum-delay multicast algorithms for mesh overlays," *IEEE/ACM Transactions on Networking*, vol. 23, 2015.

[10] X. Li *et al.*, "Scaling IP Multicast on Datacenter Topologies," in *ACM CoNEXT*, 2013.

[11] M. A. Kaafar *et al.*, "A Locating-First Approach for Scalable Overlay Multicast," in *IEEE INFOCOM*, 2006.

[12] J. Rückert *et al.*, "Software-Defined Multicast for Over-the-Top and Overlay-based Live Streaming in ISP Networks," *Journal of Network and Systems Management (JNSM)*, vol. 23, 2015.

[13] J. Rueckert *et al.*, "Flexible, Efficient, and Scalable Software-Defined Over-the-Top Multicast for ISP Environments With DynSdm," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 13, 2016.

This article has been accepted for publication in IEEE Transactions on Network and Service Management. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2023.3262294

[14] Y.-D. Lin *et al.*, "Scalable Multicasting with Multiple Shared Trees in Software Defined Networking," *Journal of Network and Computer Applications (JNCA)*, vol. 78, 2017.
[15] M. J. Reed *et al.*, "Stateless Multicast Switching in Software Defined Networks," in *IEEE International Conference on Communications (ICC)*, 2016.
[16] S.-H. Shen, "Efficient SVC Multicast Streaming for Video Conferencing With SDN Control," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 16, 2019.
[17] T. Humernbrum *et al.*, "Towards Efficient Multicast Communication in Software-Defined Networks," in *IEEE International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2016.
[18] W. K. Jia *et al.*, "A Unified Unicast and Multicast Routing and Forwarding Algorithm for Software-Defined Datacenter Networks," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 31, 2013.
[19] C. A. S. Oliveira *et al.*, "Steiner Trees and Multicast," *Mathematical Aspects of Network Routing Optimization*, vol. 53, 2011.
[20] L. H. Huang *et al.*, "Scalable and Bandwidth-Efficient Multicast for Software-Defined Networks," in *IEEE GLOBECOM*, 2014.
[21] S. Zhou *et al.*, "Cost-Efficient and Scalable Multicast Tree in Software Defined Networking," in *Algorithms and Architectures for Parallel Processing*, 2015.
[22] Z. Hu *et al.*, "Multicast Routing with Uncertain Sources in Software-Defined Network," in *IEEE/ACM International Symposium on Quality of Service (IWQoS)*, 2016.
[23] J.-R. Jiang *et al.*, "Constructing Multiple Steiner Trees for Software-Defined Networking Multicast," in *Conference on Future Internet Technologies*, 2016.
[24] B. Ren *et al.*, "The Packing Problem of Uncertain Multicasts," *Concurrency and Computation: Practice and Experience*, vol. 29, 2017.
[25] S.-H. Shen *et al.*, "Reliable Multicast Routing for Software-Defined Networks," in *IEEE INFOCOM*, 2015.
[26] A. Giorgetti *et al.*, "First Demonstration of SDN-based Bit Index Explicit Replication (BIER) Multicasting," in *IEEE European Conference on Networks and Communications (EuCNC)*, 2017.
[27] ——, "Bit Index Explicit Replication (BIER) Multicasting in Transport Networks," in *International Conference on Optical Network Design and Modeling (ONDM)*, 2017.
[28] D. Merling *et al.*, "P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast," *Journal of Network and Computer Applications (JNCA)*, vol. 169, 2020.
[29] ——, "Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4," *IEEE Access*, vol. 9, 2021.
[30] H. Chen, M. McBride, S. Lindner, M. Menth, A. Wang, G. Mishra, Y. Liu, Y. Fan, L. Liu, and X. Liu, *BIER Fast ReRoute*, https://tools.ietf.org/html/draft-ietf-bier-frr, Jul. 2022.
[31] Y. Desmouceaux *et al.*, "Reliable Multicast with B.I.E.R.," *Journal of Communications and Networks*, vol. 20, 2018.
[32] ——, "Reliable BIER with Peer Caching," *IEEE Transactions on Network and Service Management (TNSM)*, vol. 16, 2019.
[33] T. Eckert *et al.*, *Tree Engineering for Bit Index Explicit Replication (BIER-TE)*, https://datatracker.ietf.org/doc/html/draft-ietf-bier-te-arch, Jul. 2021.
[34] W. Braun *et al.*, "Performance Comparison of Resilience Mechanisms for Stateless Multicast using BIER," in *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017.
[35] G. Karypis *et al.*, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing (SISC)*, vol. 20, 1998.
[36] R. Diekmann *et al.*, "Shape-Optimized Mesh Partitioning and Load Balancing for Parallel Adaptive FEM," *Parallel Computing*, vol. 26, 2000.
[37] S. P. Lloyd, "Least Squares Quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, 1982.
[38] A. Medina *et al.*, "BRITE: An Approach to Universal Topology Generation," in *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2001.
[39] B. M. Waxman, "Routing of Multipoint Connections," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 6, 1988.
[40] F. Liu *et al.*, "The packet size distribution patterns of the typical internet applications," in *IEEE International Conference on Network Infrastructure and Digital Content*, 2012, pp. 325–332.

Daniel Merling is a Ph. D. student at the chair of communication networks of Prof. Dr. habil. Michael Menth at the Eberhard Karls University Tuebingen, Germany. There he obtained his master's degree in 2017 and afterwards, became part of the communication networks research group. His area of expertise include software-defined networking, scalability, P4, routing and resilience issues, multicast and congestion management.

Thomas Stüber is a Ph.D. student at the chair of communication networks of Prof. Dr. habil. Michael Menth at the Eberhard Karls University Tuebingen, Germany. He obtained his master's degree in 2018 and afterwards, became part of the communication networks research group. His research interests include Time-Sensitive Networking (TSN), scheduling, performance evaluation, and operations research.

Michael Menth (Senior Member, IEEE) is professor at the Department of Computer Science at the University of Tuebingen/Germany and chairholder of Communication Networks since 2010. He studied, worked, and obtained diploma (1998), PhD (2004), and habilitation (2010) degrees at the universities of Austin/Texas, Ulm/Germany, and Wuerzburg/Germany. His special interests are performance analysis and optimization of communication networks, resilience and routing issues, as well as resource and congestion management. His recent research focus is on network softwarization, in particular P4-based data plane programming, Time-Sensitive Networking (TSN), Internet of Things, and Internet protocols. Dr. Menth contributes to standardization bodies, mainly to the IETF.

2 Submitted Manuscripts (Core Content)

2.1 Minimizing Grid Electricity Consumption and On-/Off-Cyles for Heat Pumps in Single-Family Homes with PV Panels

Minimizing Grid Electricity Consumption and On-/Off-Cyles for Heat Pumps in Single-Family Homes with PV Panels

Thomas Stüber[a,*], Bernd Thomas[b], Michael Menth[a]

[a]*Department of Computer Science, University of Tübingen, Sand 13, Tübingen, 72076, Baden-Württemberg, Germany*
[b]*Reutlingen Energy Center (REZ), Reutlingen University, Alteburgstraße 150, Reutlingen, 72762, Baden-Württemberg, Germany*

Abstract

Heat pumps in single family homes provide for space heating and domestic hot water. They may be operated with self-produced electric energy from photovoltaic (PV) panels and with purchased electricity from the grid. On the one hand, heat pumps should be operated such that the consumption of purchased energy is minimized. On the other hand, a heat pump's rate of on/off cycles should be minimized to maximize its lifetime. This calls for optimized operation schedules for heat pumps.

In this work, we model this optimization problem as mixed integer linear program (MILP) including physical constraints of the heat pump. Energy can be stored over time by hot water storage and the thermal capacity of the building. When the hot water storage is getting discharged, an electric heater helps to quickly refill it. As input data we utilize measured traces for energy production and consumption. In a first step, the schedule of the heat pump is optimized for an entire year. However, future production and demands are not known in advance, which affects the realistic optimization potential. Therefore, we introduce incremental optimization over time using a rolling horizon approach and adapt it such that forecast can be used instead of known data. The paper provides extensive numerical results for a case study. A prominent result is that optimized schedules can significantly reduce both purchased energy from the grid and the rate of on/off cycles, even with simple

*Corresponding author
Email address: thomas.stueber@uni-tuebingen.de (Thomas Stüber)

forecasts. Moreover, heat pumps with power modulation can better achieve these goals than heat pumps without power modulation.

Keywords: Heat pump, Scheduling, Space heating, DHW, Optimization, Forecasting, Rolling horizon, Multi-objective optimization

---

1. Introduction

There is a trend to more photovoltaic (PV) panels on rooftops of single-family homes. However, selling PV energy when available and purchasing energy when needed is less attractive for private persons compared to utilizing the PV energy directly onsite. This follows from the tariffs. In Germany a net price of about 33.6 ct/kWh has to be paid for electricity drawn from the grid. The feed-in tariff of PV electricity is currently at about 8.2 ct/kWh.

Heat pumps leverage electrical energy to take advantage of low temperatures to produce heat for domestic hot water (DHW) and space heating. They avoid fossil fuels and CO_2 emissions if electricity stems from regenerative energy resources. Moreover, they may be partly operated with PV energy from the own rooftop. In addition, they may be combined with a moderately sized heat storage. The heat storage consists of an insulated reservoir for DHW and the mass of the house allowing the indoor temperature to vary in a fixed range. However, two challenges remain. First, PV energy does not always suffice for heating, in particular during winter, so that additional grid energy is needed. Second, heat pumps must not be switched on and off frequently as this reduces their lifetime, makes maintenance contracts more expensive, and renders their internal physical processes less efficient. Thus, purchased grid energy should be minimized while keeping the number of on/off cycles low. This poses an interesting optimization problem with two opposing goals, a so-called Pareto-optimization problem. That is, many schedules exist such that there is no better schedule for grid energy with the same number of on/off cycles or vice-versa. In practice, a Pareto-optimal schedule needs to be chosen that avoids both excessive grid energy and a large number of on/off cycles.

We model the optimization problem respecting time-dependent heat storage losses and temperature-dependent efficiency of the heat pump. Input data are historic heat consumption, outside temperature, and PV energy production curves for a specific single family home. In practice, forecasts for energy consumption and production must be taken as a planning basis

instead of historic data. This creates several challenges. Since reasonable forecasts are available only for a few days, schedules can be computed only for a few days, too. As the objective is to optimize a longer period, we leverage a so-called rolling horizon approach [1]. Another challenge is that forecasts may be inaccurate. Production may be overestimated or consumption may be underestimated. Both errors may cause premature heat storage depletion so that heat demands cannot be fulfilled in time. We introduce a concept to reduce the likelihood for such events. Finally, we investigate the impact of PV panel sizes and heat storage sizes on potential savings of purchased electricity and on on/off cycles of the heat pump. Heat pumps with and without power modulation are considered.

This paper is structured as follows. Section 2 discusses related work. In Section 3 we describe physical aspects of the considered heat pump, the storage for domestic hot water and space heating, and we model the optimization problem by a mixed integer linear program (MILP). Section 5 utilizes the MILP to optimize schedules with various constraints. In particular, a rolling horizon strategy is developed, the impact of PV peak-power, heat storage capacities, and the use of forecasts instead of historic data are investigated. Heat pumps with and without power modulation are studied and their behaviour over time is analyzed. Section 6 summarizes this work and draws conclusions.

2. Related Work

In this section we give a short overview of research works concerning the control of heat pumps.

In [2] and [3], simulation models in MATLAB and TRNSYS are used to investigate the potential of an appropriate control for running heat pumps in combination with thermal energy storages (TES) more efficiently and sustainably. It is mentioned that either the TES can be a hot water storage tank, or the building itself can serve as a thermal energy storage as, e.g., thermal capacity of floors and walls can be utilized in this respect.

A common control method for heat pumps is model-predictive control (MPC) as presented in [4] and [5]. It is reported that MPC strategies are confirmed with respect to a heat pump installed in a climatic chamber, in order to simulate real outdoor conditions at the test stand. In addition, MPC strategies for systems consisting of heat pump and solar thermal in combination with a TES are presented and verified by simulations over a

period of 24 hours. In both cases it is shown that the investigated systems work well. On the one hand the power consumption of the heat pump does not exceed a defined limit value. On the other hand, the MPC contributes to a reduction of operating costs by 1% to 7% and of CO_2 emissions by 3% to 17%.

In [6] and [7], the impact of heat pumps on the electricity grid is investigated using various simulation models. Analyses of the integration of microgrids are carried out in detail by means of simulations for a system with PV, thermal energy storage, and heat pumps. Basically, the heat pumps in combination with TES are intended to shift the exchange of electricity between the building and the grid. The analysis reveals parameters that facilitate an improved economic operation of the microgrid system. In addition, indicators are introduced for evaluating the performance of control mechanisms for passive buildings on the electricity grid without simulating the grid itself. The modelling of the simulation was carried out in Modelica.

Investigations regarding the coefficient of performance (COP) of heat pumps can be found in [8] and [9]. One question answered is how exact the calculation of a building's heat demand needs to be for a proper design of a heat pump. For this purpose, reference values are determined using a steady state wall model without taking the thermal capacity of the wall into account. Based on average daily and monthly temperatures these results are compared to solutions of partial differential equations for heat losses and heat consumption with respect to actual temperature profiles. It is shown that the error of the calculation for heat pump COP is in the range of only 2.5%. In [9] the simulation of the COP for an air source heat pump for generation of domestic hot water is reported. The relevant input data are taken from experiments where data from both the heat pump circuit and the domestic hot water installation were recorded. Two scenarios are compared in the simulation. First, no domestic hot water is drawn during operation of the heat pump; second, domestic hot water is drawn at any time, independently of the operation of the heat pump. The simulation results show that the COP in both cases is on average above the value of 2 without significant difference.

The effect of different flexibilities on the heat pump's periods of operation is investigated in several studies, e.g., [10], [11]. One approach is to increase the charging temperature of a hot water tank to create flexibility for the operating hours of the heat pump. Another option is imposing price signals as an incentive to operate the heat pump at the lowest possible cost while taking advantage of the flexibility. The simulations show that heat pumps

in combination with TES have the potential to shift loads over a period of several hours. In addition, the thermal inertia of the building can be utilized to control the periods of operation of heat pumps. This effect is studied in detail by means of building simulations in [11], and the results are compared to measurements from a small apartment building. It is shown that a high thermal capacity is needed to shift the operating hours times of the heat pump to daytime hours in order to enable operation on PV electricity, only.

3. Model

The following section gives an overview of the components of the considered case study. First, we describe the components of the evaluation scenario and their setup. Then, we discuss the time series data sets used.

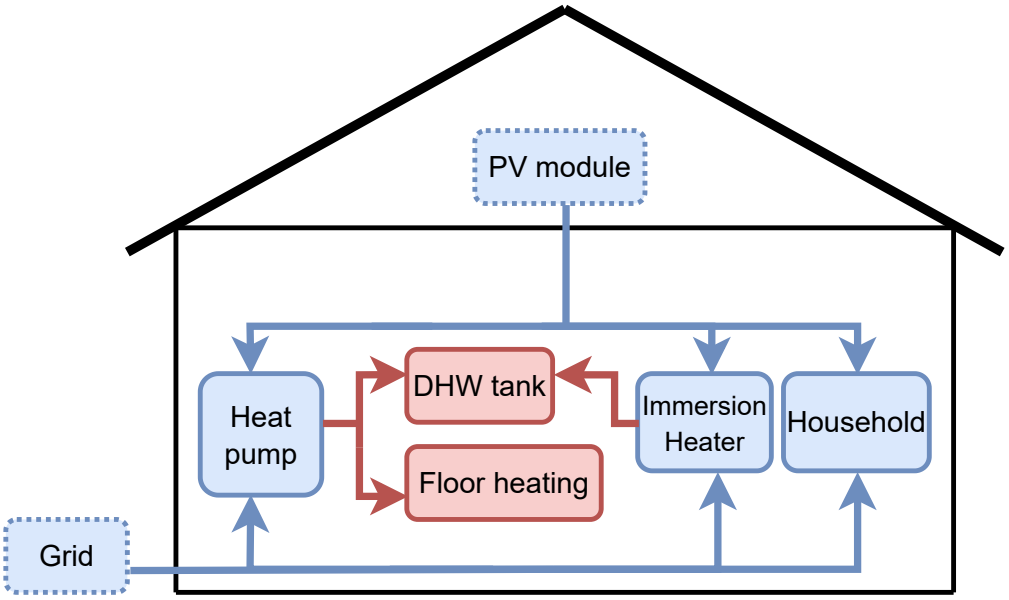


Figure 1: 1-Zone building model: arrows indicate flows of electrical energy (blue) and thermal energy (red).

3.1. Model Description and Analysis

In our case study we consider a residential building which is equipped with a heat pump and a PV-module as depicted in Figure 1. The heat pump has

two supply feeds: one for serving the heating system for residential rooms and one for preparing DHW. Only one supply feed can be used at any given time so that the heat pump cannot serve the heating system and produce DHW simultaneously. A brine-water heat pump is utilized with 10.8 kW maximal thermal power at standard point B0/W35. In general, the electrical and thermal power of the heat pump depend on the outside temperature. The electrical power can be modulated continuously between 30% and 100% of the respective maximal electrical power. We also consider an almost identical heat pump in on/off-mode, i.e., it can be operated only with 0% or 100% of the maximal thermal power. Additionally, an electrical immersion heater is installed, which may be used for DHW generation when the reservoir is depleted. The heat for residential rooms is stored in the building mass, the produced DHW in a hot water tank. We consider the temperature of the building and the temperature of the water in the DHW tank as measure for the amount of stored energy. The amount of energy stored in the building mass is restricted to the interval 40 kWh to 89 kWh, leading to a capacity of the thermal energy storage of 49 kWh. DHW storage is restricted to the interval 11 kWh to 15.65 kWh, which correspond to water temperatures of 10 °C and 50 °C, leading to a capacity of the thermal energy storage for DHW of 4.65 kWh. Both thermal storages are subject to thermal loss over time, which are specified in the paragraph below. In addition to the electrical demand of the heat pump and the immersion heater there is also an electrical demand for the domestic consumers in the household. The demands of all electrical consumers can be partially fulfilled by a PV-module on the roof of the building. The residual demands are satisfied by an energy supplier via the grid. All parameters of heat pump, PV-module, immersion heater, and thermal and DHW storage are compiled in Table 1.

3.2. Thermal and Electrical Model

Thermal energy can be stored in the floor heating system (heat) or the hot water tank (DHW). Demand for thermal energy from the floor heating system corresponds to heating up residential rooms. Demand for thermal energy from the DHW storage corresponds to the delivery of hot water. The stored thermal energy in both storages is subject to thermal decay, i.e., the thermal energy is lost due to imperfect thermal insulation. We model this loss as a linear decay which depends on the state of charge (SOC) of the respective storage, since SOC is assumed directly proportional to storage temperature as outlined before. The heat storage is subject to a loss of $L^h_{\max} := 0.0784\,\mathrm{kWh/h}$

Table 1: Model parameters for heat pump and PV modules.

Parameter	Value
Heat pump type	Air source (air-water) heat pump
Heat pump nominal thermal power	$10.8\,\mathrm{kW_{th}}$
Heat pump COP	4.6 (at B0/W35)
Heat pump modulation	30%-100%
Minimal operation time	10 min
Minimum rest time	5 min
PV peak power	5 kW

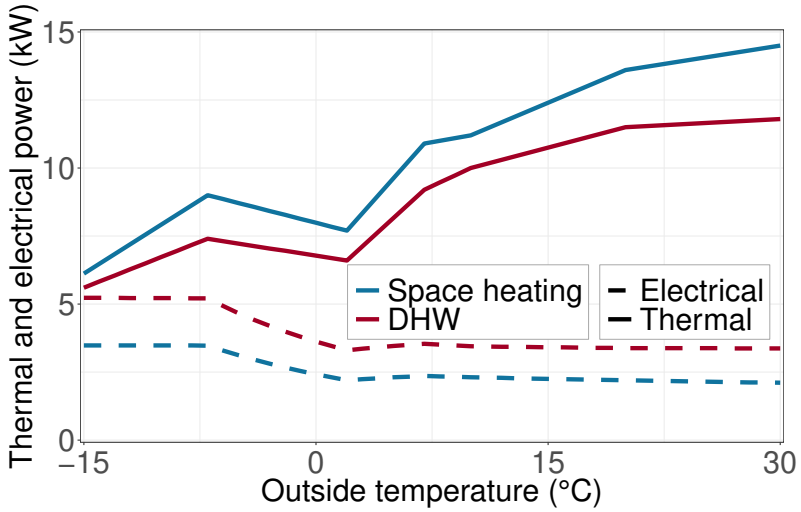
when completely charged and $L^h_{\min} := 0.029\,\mathrm{kWh/h}$ when discharged. The DHW storage is subject to $L^w_{\max} := 0.00743\,\mathrm{kWh/h}$ when completely charged and $L^w_{\min} := 0.00186\,\mathrm{kWh/h}$ when discharged. Losses are linearly interpolated between the respective extreme values for other SOCs.

The maximum thermal and electrical power of the heat pump depend on the outside temperature and the supply temperature. Figure 2(a) depicts these relationships. The electrical power ranges between 2.11 and 3.48 kW for space heating and between 3.30 and 5.23 for DHW. The resulting thermal power ranges between 6.12 and 14.50 kW for space heating and between 5.60 and 11.80 kW for DHW. The ratio of the thermal and electrical power is the coefficient of performance (COP). The electrical and thermal power of the immersion heater is 5 kW, i.e., it has a COP of 1.

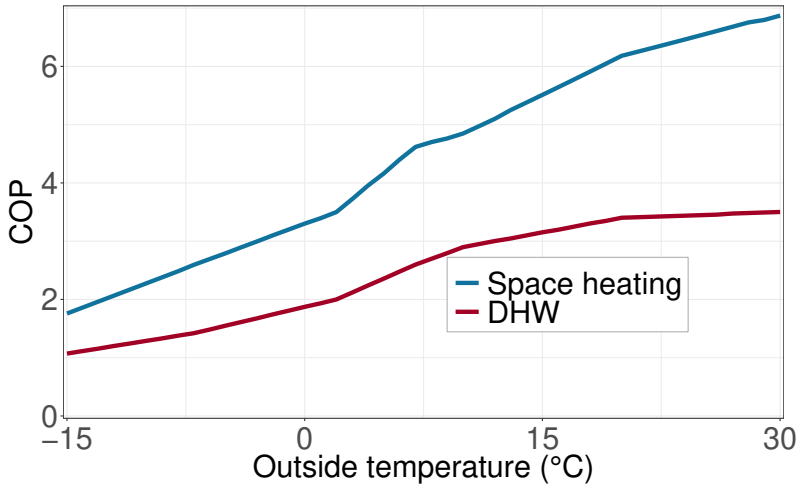
3.3. Demand and Supply Visualization over Time

We illustrate how energy production and consumption vary over a year. We use time series for outside temperature, PV-generation, heat demand, DHW demand, and domestic electricity demand from a residential building in Düsseldorf, Germany. The data set was collected for the year 2018. The resolution of all time series is 1 min.

Table 2 briefly summarizes important properties of the data set. Space heating required 9908 kWh and domestic hot water 2063 kWh. To satisfy these demands, a heat pump requires at least 2602 kWh electrical energy. The electrical demand for the household itself is 3985 kWh so that in sum 6587 kWh electrical energy were required. We accumulated the household demand for electrical energy that exceeds the PV energy per quarter-hour



(a) Thermal and electrical power.



(b) Coefficient of Performance (COP) of the heat pump.

Figure 2: Electrical and thermal characteristics of the heat pump for the heat and DHW supply feed.

and obtained 2340 kWh. That means, out of the 3985 kWh at least 2340 kWh cannot be supplied by PV. However, this is only a lower bound since the demand for electrical power by the household may exceed the electrical

Table 2: Model parameters for energy demands and storage in the case study.

Parameter	Value
Evaluation period	1 year
Demand for space heating	9908 kWh
Demand for DHW	2063 kWh
Demand for electrical energy (w/o heat pump)	3985 kWh
Building type	Single family house in Düsseldorf, Germany
Min./Max. outside temperature	-13.3 °C/33.3 °C
Yearly average outside temperature	11.4 °C
Capacity of TES for space heating	49 kWh
Volume of TES for DHW	100 l
Capacity of TES for DHW	4.65 kWh
Electric heater for DHW	5 kW

supply by the PV even though the supplied energy exceeds the demanded energy within a quarter-hour.

The PV panels generated 6584 kWh electrical energy per year. Thus, the yearly production and consumption (6587 kWh) of electrical energy were almost equal. However, electrical energy is not always produced when needed to satisfy the electrical demands. Figure 3 shows the production and consumption of electrical energy over a year. Demand for heat is converted to electrical demand using a COP of 4.6. We remark that this COP is arbitrary and chosen only for visualization. The figure shows that during winter, the demand for electrical energy clearly surpasses its provision while there is significantly more electrical energy supply than demand during summer. This is due to the large demand of thermal energy in winter and the large production of PV energy in summer. In theory, a battery could be used to store electrical energy generated in summer for consumption in winter. However, that battery must be large. As such a large battery is also costly, we renounce on it in our evaluation.

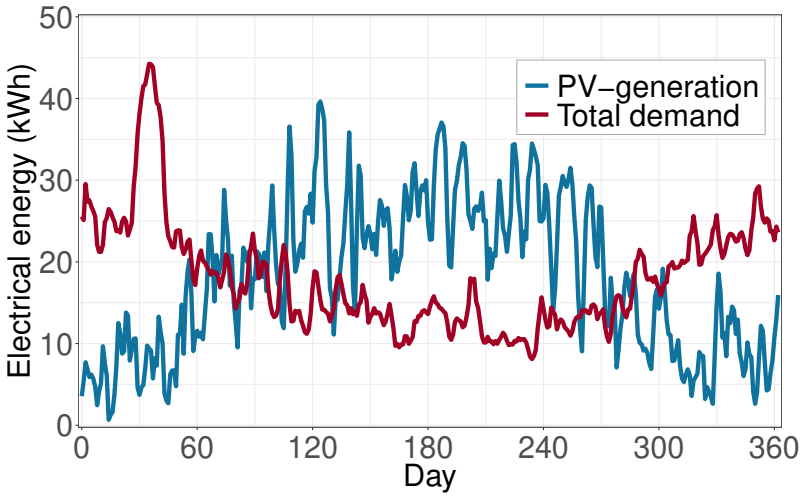


Figure 3: Production and consumption of electrical energy over a year. Consecutive per-day data points are connected for better visibility.

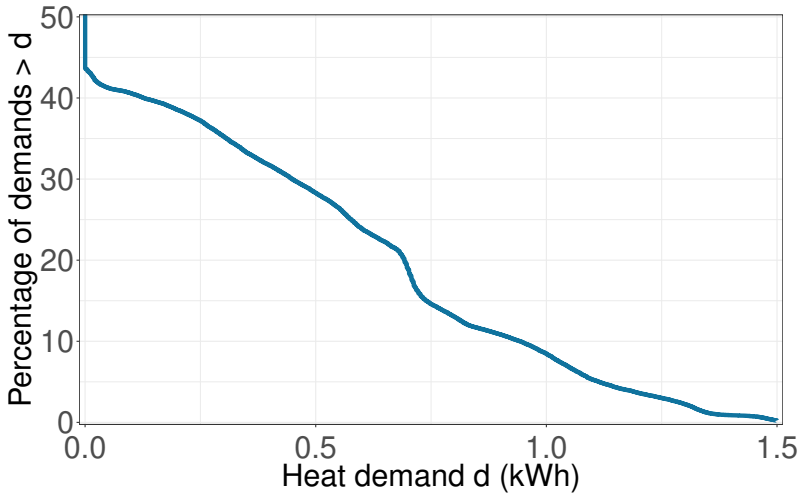
3.4. Demand Distribution

Thermal demand is required for space heating and domestic hot water. We break the yearly demands down to quarter-hours and present their complementary cumulative distribution function (CCDFs) in Figures 4(a) and 4(b). In 57% of all quarter-hours of a year, no energy is needed for space heating while space heating almost never requires more than 1.5 kWh per quarter-hour. In 80% of all quarter-hours, no hot water is requested. In 99% of all quarter-hours less than 1.25 kWh thermal energy is needed for DHW. However, in rare cases (100 out of 35040 quarter-hours per year) 1.97 kWh and more are needed. In 4 exceptional quarter-hours, the demand for DHW surpassed the capacity of the water tank. The maximum demand was 6.09 kWh. This could be achieved when the storage was quickly depleted and heat pump as well as electric immersion heater are needed to heat up the storage again.

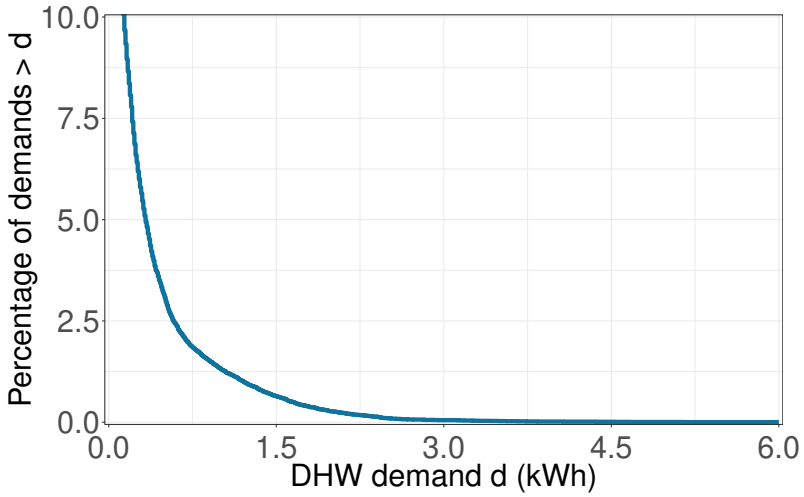
An example for such a case is when multiple persons have a shower in different bathrooms.

4. A MILP for Schedule Optimization

We describe a MILP model of the heat pump optimization problem to find the optimal schedule for the heat pump (and the immersion heater) in



(a) Space heating.



(b) Domestic hot water.

Figure 4: CCDFs of thermal energy demand in 2018 on a quarter-hour basis.

the evaluation scenario. We use this model later to compute energy demands and other properties in various evaluation settings. First, we introduce fundamentals about MILPs. Then, we define the notion of a heat pump schedule and its components. Afterwards, we discuss modelling assumptions used to

transform the scheduling problem into a MILP. Finally, we state and elaborate on the variables and constraints of the MILP model.

4.1. Fundamentals of MILPs

A MILP describes the set of valid solutions to a problem by linear inequalities on a set of variables. The variables can take values from discrete or continuous sets. The linear inequalities restrict the set of all possible variable assignments to those assignments which correspond to valid solutions of the modeled problem. Each variable assignment which fulfills all inequalities is a feasible solution to the MILP and vice versa. A linear objective function is used to compare feasible solutions. Given some feasible solution, the solution is denoted as optimal if no feasible solution with lower objective value exists.

An MILP solver can compute the optimal solution for arbitrary MILPs. During the solving process, lower and upper bounds for the objective value of the optimal solution are inferred. The optimal solution is found when upper and lower bound meet.

4.2. Schedules

We denote the time interval for which the heat pump's schedule is planned as planning horizon. In the following, all times are assumed to be relative to the start of the planning horizon. The schedule of a heat pump is a list of time intervals in which the heat pump runs and a mapping of time instants to heat pump behavior, e.g., which supply feed is used. Additionally, a schedule contains a mapping of time instants in which the heat pump runs with power modulation specified by the modulation coefficient. The modulation coefficient at a time instant t is the fraction of the actual thermal power of the heat pump and its maximum thermal power with respect to the outside temperature at time t. The schedule for the immersion heater is also a list of time intervals in which the heater is in operation. However, the immersion heater has only one mode of operation and cannot be modulated.

Scheduling in the context of this work is the process of finding a valid schedule for the heat pump and the immersion heater.

4.3. General Assumptions

Let h be the length in time of the planning horizon. We divide the planning horizon in discrete time slots with length l. In the remainder of this work, we assume $l := 15\,min$. Thus, the number of time slots is $n := \frac{h}{15\,min}$.

Time slots are denoted by their index in the range $[1, n]$. The heat pump can either run or pause an entire time slot. The power modulation and the supply feed can only be changed at the beginning of a time slot.

4.4. Variables and Constraints

Variables are used to capture a complete description of a schedule for the heat pump and the immersion heater. Constraints restrict the set of variable assignments to assignments that correspond to valid schedules. This section makes the variables and constraints of the MILP model explicit. The variables and parameters of the model are compiled in Tables 3 and 4.

Table 3: Variables of the MILP model.

Variable	Type	Definition
P_t^h, P_t^w	binary	1 if the heat pump produces space heating/DHW in time slot t, 0 otherwise
M_t^h, M_t^w	continuous	Modulation factor of the heat pump in space heating or DHW mode in time slot t
I_t	binary	1 if the immersion heater is on in time slot t, 0 otherwise
G_t, PV_t	continuous	Consumed power from grid and PV-modules in time slot t
H_t, W_t	continuous	State of charge (SOC) of heat and DHW storage after time slot t
S_t	binary	1 if the heat pump starts running in time slot t, 0 otherwise

4.4.1. Supply Feed

Let t be a time slot of the planning horizon. The binary variables P_t^h and P_t^w capture whether the heat pump runs in the time slot t. Thermal power is produced for the space heating supply feed if and only if P_t^h is set to 1. Analogously, thermal power is produced for the DHW supply feed if and only if P_t^w is set to 1. The heat pump can only produce for at most one supply feed per time slot. This is enforced by the following equation:

$$P_t^h + P_t^w \leq 1. \tag{1}$$

Table 4: Parameters of MILP model.

Parameter	Definition
d_t, h_t, e_t	Demand for DHW, space heating and electrical energy in time slot t
th_t^h, th_t^w	Thermal power of the heat pump in time slot t (depends on the outside temperature)
el_t^h, el_t^w	Electrical power of the heat pump in time slot t (depends on the outside temperature)
el^i, th^i	Electrical and thermal power of the immersion heater (both 5 kW in the evaluation scenario)
H_0, W_0	State of charge (SOC) of space heating and DHW storage at the beginning of the planning horizon
$H_{\min}, H_{\max}, W_{\min}, W_{\max}$	Minimal and maximal state of charge (SOC) of space heating and DHW storage
$L_{\min}^h, L_{\max}^h, L_{\min}^w, L_{\max}^w$	Minimal and maximal thermal loss of space heating and DHW storage
p_t	Available PV power in time slot t
l	Length of a time slot
n	Number of time slots
a, b	Weights for grid energy demand and heat pump starts in the objective function

4.4.2. Power Modulation

The heat pump can modulate its thermal power, i.e., it can work with less thermal power than its maximum thermal power. The modulation coefficient must be in the range $[0.3, 1]$ when the heat pump is running and 0 otherwise. For every time slot t, we use variables M_t^h and M_t^w to capture the modulation coefficient for space heating and DHW generation, respectively. Equation 2 and 3 enforce that the variables for the modulation coefficient are at most 1 when the heat pump runs the respective mode and 0 otherwise.

$$M_t^h \leq P_t^h \tag{2}$$

$$M_t^w \leq P_t^w \tag{3}$$

Additionally, Equations 4 and 5 model that the variables for the modulation coefficient is at least 0.3 when the heat pump runs in the respective mode.

$$P_t^h - 1 \leq M_t^h - 0.3 \tag{4}$$
$$P_t^w - 1 \leq M_t^w - 0.3 \tag{5}$$

4.4.3. PV Power

The used PV power PV_t during time slot t must be less than the available PV power during this time slot p_t.

$$PV_t \leq p_t \tag{6}$$

4.4.4. Immersion Heater

For every time slot t, we introduce a binary variable I_t. The immersion heater is in operation during t if and only if I_t is set to 1.

4.4.5. Generation and Demand

Let el_t^h and el_t^w be the maximum electrical power of the heat pump during time slot t for space heating and DHW mode, respectively. These parameters depend on the outside temperature according to the characteristics in Figure 2(a). Furthermore, let e_t be the domestic electrical demand during t. We introduce a continuous variable G_t to capture the power taken from the grid during t. The power consumed by heat pump, immersion heater, and domestic electrical demand, must equal the combined power from the PV-module and the grid. This is enforced by the following equation:

$$M_t^h \cdot el_t^h + M_t^w \cdot el_t^w + el^i \cdot I_t + e_t = PV_t + G_t. \tag{7}$$

4.4.6. Thermal Storage

We introduce continuous variables H_t and W_t for the thermal energy stored in the floor heating system and the DHW storage after time slot t. Furthermore, we define H_0 and W_0 to be the initial thermal energy contained in both storages at the start of the planning horizon. The SOC of a storage after a time slot t can be computed with its SOC after the previous slot $t-1$, and the thermal power of the heat pump in this time slot for the respective supply feed, minus the thermal loss of the storage, and the respective demand during the time slot. Additionally, the immersion heater with thermal power th^i must be considered for the SOC of the DHW storage. The thermal energy generated by the heat pump and the heater in a time slot can be computed

by the employed thermal power in this time slot multiplied with the length of a time slot l. Let h_t and d_t be the demands for space heating and DHW in time slot t, respectively. The following equations enforce the intended semantics:

$$H_t = H_{t-1} - l \cdot \left(L^h_{\min} + (L^h_{\max} - L^h_{\min}) \frac{H_{t-1}}{H_{\max} - H_{\min}} \right) + l \cdot th^h_{t-1} \cdot M^h_{t-1} - h_{t-1} \tag{8}$$

$$W_t = W_{t-1} - l \cdot \left(L^w_{\min} + (L^w_{\max} - L^w_{\min}) \frac{W_{t-1}}{W_{\max} - W_{\min}} \right) + l \cdot th^w_{t-1} \cdot M^w_{t-1} + l \cdot th^i \cdot I_{t-1} - d_{t-1}. \tag{9}$$

We remark that the terms in the parenthesis correspond to linear interpolation of the thermal losses.

The SOC of the heat storage for space heating must range between its minimum value $H_{\min}$ and its maximum value minimum value $H_{\max}$. Similarly, the SOC of the DHW storage must range between $W_{\min}$ and $H_{\max}$. Both of these constraints are enforced by the following inequalities for all time slots t:

$$H_{\min} \leq H_t \leq H_{\max} \tag{10}$$

$$W_{\min} \leq W_t \leq W_{\max}. \tag{11}$$

When working with historic input data, $H_{\min}$ and $W_{\min}$ are zero. When working with forecast input data, $H_{\min}$ and $W_{\min}$ are set to the respective recharge threshold (cf. Section 5.3.3).

4.4.7. On/Off Cycles

We define the start of the heat pump for time slot t if it was not running in time slot $t-1$ and it runs in time slot t. We introduce a binary variable S_t which should be set to 1 if and only if a start of the heat pump occurs in time slot t. The binary variables for the heat pump $P^h_{t-1}, P^w_{t-1}, P^h_t, P^w_t$ can be used to express the intended semantic for all time slots $t > 1$. Equation 12 enforces that the definition of a start in time slot t implies that S_t is set to 1. Equations 13 and 14 enforce the opposite direction of this implication.

$$(1 - P^h_{t-1} - P^w_{t-1}) + (P^h_t + P^w_t) - 1 \leq S_t \tag{12}$$

$$S_t \leq 1 - P^h_{t-1} - P^w_{t-1} \tag{13}$$

$$S_t \leq P^h_{t-1} + P^w_{t-1} \tag{14}$$

For the first time slot $t = 1$, there is no previous time slot. Thus, the definition of a start is slightly modified. The variable S_1 is set to 1 if and only if the heat pump runs in the first time slot. This can be enforced with the following equation:

$$P_0^h + P_0^w = S_0. \tag{15}$$

For certain cases in the evaluation, the heat pump may already be working at the beginning of the planning horizon from a previous planning horizon. In this case, Equation 15 is substituted with:

$$S_0 = 0. \tag{16}$$

4.4.8. Minimum Running Time

The variables for on/off cycles can be employed to enforce minimum running times of the heat pump, in order to ensure that the heat pump is at least a certain time in operation after starting and to avoid by this means any damages of the heat pump by very frequent on/off-cycles. Let k be the minimum number of time slots the heat pump must be in operation after an on/off cycle started. The following equation enforces the implication that the heat pump must be in operation for k time slots after it has been started.

$$k \cdot S_t \leq \sum_{i=t}^{\min(n,t+k-1)} P_i^h + P_i^w \tag{17}$$

4.4.9. Pauses

Pause times can be enforced in a similar way as minimum running times. If the heat pump must pause for at least k time slots before running again after it has been stopped, the following equality must hold.

$$k \cdot S_t \leq \sum_{i=\max(1,t-k)}^{t-1} 1 - P_i^h - P_i^w \tag{18}$$

Minimum running times and pauses are shorter in case the heat pump was running shortly before the planning horizon in a previous schedule. For instance, assume the minimum running time of the heat pump is three time slots and the heat pump is in operation during the last time slot of a planning horizon. In this case, the heat pump must be in operation for at least two time slots at the beginning of the next planning horizon. We remark that trivial modifications to Equations 17 and 18 are necessary in such cases.

4.4.10. Objective

The objective function depends on the respective evaluation and is a linear weighted sum of the consumed energy from the grid and the number of heat pump on/off cycles. We assume the weight of the consumed grid energy to be one. The weight of the number of on/off cycles is denoted as C. The weighted sum should be minimized in all evaluation scenarios and is given in the following:

$$min : \sum_{t=1}^{n} l \cdot G_t + C \cdot \sum_{t=1}^{n} S_t. \tag{19}$$

5. Optimization Results

In this section, we incrementally develop a strategy for optimization of heat pump schedules using simple forecast for energy production and consumption.

We first optimize the schedule for the heat pump for an entire year (1.1.2018 – 31.12.2018) in one shot based on exact energy production and consumption data. As this is an unrealistic assumption, we propose continuous optimization with 3-days-ahead knowledge of energy production and consumption using a rolling horizon approach. Moreover, we propose how to minimize both purchased grid energy and number of needed on/off cycles. Afterwards, we substitute the known energy production and consumption data by simple forecasts and introduce a mechanism to cope with estimation errors. We then evaluate the impact of system parameters on required grid energy and on/off cycles and compare the performance of optimized operation with the one of a simple operation method. Finally, we analyze the operation of heat pumps running on optimized schedules.

Throughout this section, schedules are computed with the MILP of Section 3 based on different optimization strategies with different planning horizons, empirical or forecast input data, and using adapted objective functions. The schedules are always executed on empirical data for performance evaluation purposes. Thus, computing schedules based on forecast data may cause premature storage exhaustion or longer lasting energy in the storage. This effect is handled by the proposed mechanism to cope with estimation errors.

As the MILP solver takes too long for the computation of optimum values, the computation is stopped when found solutions are at most 0.1% close to the optimum. As finding such a solution or a valid solution at all takes

too much time, the computation is also stopped after 5 minutes or 24 hours depending on the specific experiment.

5.1. Optimization of an Entire Year in a Single Run

We compute lower bounds on required grid energy and on/off cycles, respectively. To that end, we take historic energy production and consumption data as input. We formulate the optimization problem for an entire year within a single MILP, i.e., the input data are all known in advance. The ILP solver is stopped after 24 hours if sufficiently accurate results have not been found before. The results of this subsection are compiled in Table 5.

5.1.1. Minimization of Grid Energy

In the first experiment series, we utilize the MILP to minimize the purchased grid energy for a heat pump with and without modulation. That means, C is set to zero in the objective function in Equation (19). Thus, the number of on/off cycles is not part of the objective function.

For a heat pump without modulation, the ILP solver could not find a solution, but provided a lower bound of 4011 kWh purchased grid energy.

The MILP models for all experiments in Section 5.1 consist of 385439 inequalities with 350400 variables, 140160 of them are restricted to integer values. This is very large and makes most of these problems unfeasible within acceptable time.

For a heat pump with modulation the ILP solver found 3897 kWh purchased grid energy as near-optimum solution. It can be achieved with 600 on/off cycles.

5.1.2. Minimization of On/Off Cycles

In the third experiment series, we utilize the MILP to minimize the number of on/off cycles needed for a valid schedule for a heat pump with and without modulation. That means, l is set to zero and C is set to 1 in the objective function in Equation (19). Thus, the purchased grid energy is not part of the objective function. Here, the ILP solver could not even find a valid solution to provide an upper bound for the number of needed on-/off cycles. Nevertheless, it could infer a lower bound, which is 3 for heat pumps without modulation and 75 for heat pumps with modulation. However, this information is not useful.

Table 5: Required grid energy and on/off cycles for optimized heat pump schedules. (single: single optimization of a year, RH: RH optimization of a year

Variant	W/o modulation		W/ modulation	
	Grid energy (kWh)	On/off cycles	Grid energy (kWh)	On/off cycles
single/e↓	≥ 4011	–	3897.47	600
single/c↓	–	≥ 3	–	≥ 75
RH/e↓	4078.91	1332	3901.59	1458
RH/c↓	7975.60	256	7635.27	194
RH/weighted sum, C=5 kWh	4478.52	239	4239.23	150

5.2. Rolling Horizon: Stepwise Optimization over Time

In practice, only forecast energy production and consumption data serve as input to the optimization problem instead of historic data from 2018. Such forecast data data are available only a few days in advance so that a schedule cannot be optimized for an entire year in advance. In the following, we adopt the well-known optimization strategy "Rolling Horizon" (RH) to cope with that problem. RH is a well-known strategy for long-term optimization problems based on only limited forecasts [1]. We apply it using the MILP from 4 with a planning horizon of 3 days. That means, we use the current system state and historic energy production and consumption data of the last 3 days to compute optimized schedules for the next 3 days. On the basis of the obtained schedule, we calculate the new system state after one day and repeat a 3-days optimization for that time. We repeat this procedure until the end of the year is reached.

As this optimization strategy requires only schedule optimization for 3 days, the corresponding MILPs consist of only 3166 inequalities with 2880 variables, 1152 of them being binary. This leads to significantly less complexity than the MILPs for the optimization of an entire year in Section 5.1. However, 363 consecutive schedules are needed to compose an overall schedule for an entire year. Therefore, we limit the computation time for the ILP solver to 5 minutes instead of 24 hours for the remainder of this work.

The results for the following experiment series are also compiled in Table 5. We first minimize only the required grid energy. This yields schedules requiring 4078.91 kWh grid energy and 1332 on/off cycles for heat pumps

without modulation and 3901.59 kWh grid energy and 1458 on/off cycles for heat pumps with modulation. These values for grid energy are close to the yearly upper bound of 3998 kWh and the optimum of 3897.47 kWh, respectively. However, the numbers of on/off cycles are 1332 and 1458. They are significantly higher than in the one-shot optimization for an entire year, which is obviously caused by the simple application of the RH method.

We now minimize only the number of on/off cycles. In contrast to Section 5.1, optimium solutions are found due to the short planning horizons of 3 days. We obtain schedules with 194 and 256 on/off cycles for heat pumps without and with modulation, respectively. However, they require 7635 kWh and 7976 kWh grid energy. These numbers are very large compared to the minimum required grid energy.

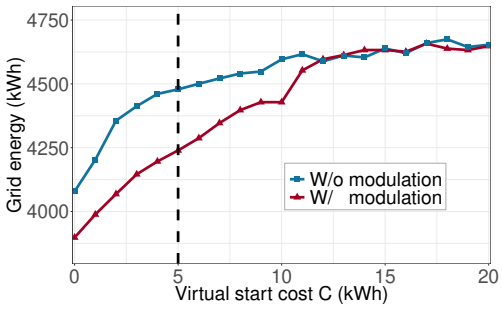
Thus, the simple adoption of the RH method allows to effectively minimize either the required grid energy or the number of on/off cycles, but not both.

5.2.1. Pareto Optimization Using Weighted-Sum

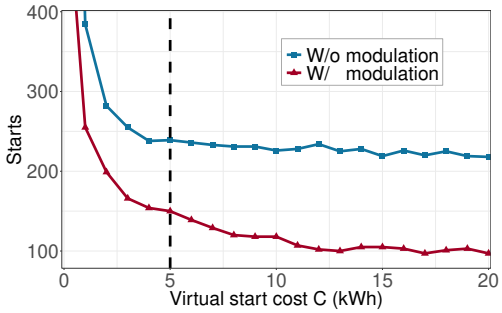
So far, we computed schedules leading to a minimum purchased grid energy or to a minimum number of on-/off-cycles. Especially with the simple application of the RH method, the respective other metric is significantly degraded. However, schedules are desired that lead to only reasonably little purchased grid energy while causing only a reasonably few on-/off cycles. This is a multi-objective optimization problem and, more specifically, a Pareto-optimization problem as optimizing one metric degrades the other.

Weighted-sum optimization [12] is a well-known strategy to cope with Pareto optimization problems. Its objective function consists of a weighted sum of both metrics (cf. Equation 19) with $l = 1$ and C a chosen positive parameter. The parameter C is essentially a virtual start cost in kWh which is added per start to the purchased grid energy for a schedule, leading to the objective function to be minimized. That effects that additional starts are allowed if they sufficiently reduce the purchased grid energy. Of course, the virtual start costs do not contribute to the real costs of purchased energy.

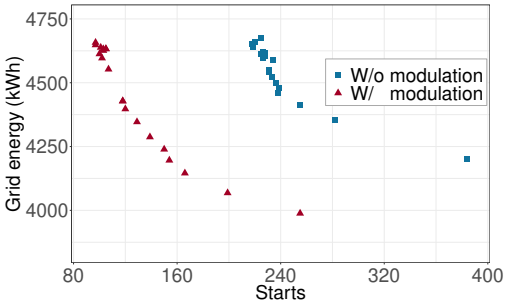
We apply this weighted-sum optimization for heat pumps without and with modulation with different virtual start costs C. The results are reported in Figures 5(a) and 5(b). With increasing virtual start cost C, the energy required from the grid increases and the number of starts decreases. This holds for heat pumps with and without modulation.



(a) Required grid energy depending on virtual start cost C.



(b) Required on/off cycles depending on virtual start cost C.



(c) Scatter plot of required grid energy and on/off cycles for weighted-sum-optimized schedules. The linked points approximate a Pareto front.

Figure 5: Schedules for heat pumps without and with modulation are computed with rolling horizon and weighted-sum optimization. Historic production and consumption data are used as optimization input.

The figures also show that optimized schedules for heat pumps with modulation require less grid energy and fewer starts than those for heat pumps without modulation. This can be explained a follows. Heat pumps without modulation are either on or off. During times with only little PV supply, they require additional grid energy when running. In contrast, heat pumps with modulation can be operated at a lower power level so that less grid energy is needed during times of only little PV supply. As a consequence, the heat pumps' on-cycles are longer, which automatically reduces the number of on/off cycles.

While the required grid energy increases approximately linearly with virtual start costs, the number of starts decreases first quickly and then slowly. Increasing the start cost C beyond 5 kWh further increases the required grid energy, but it does not effectively reduce the number of starts. Therefore, we take a virtual start cost of $C = 5$ kWh as preferred value for studies in the remainder of this work. This is indicated by a dashed vertical line in Figures 5(a) and 5(b).

We now select schedules for which no other schedule with less grid energy and less starts exist. They approximate a Pareto-optimal set and are compiled in Figure 5(c). The x-axis indicates the number of starts needed by a schedule and the y-axis shows its required grid energy. The resulting lines constitute approximate Pareto fronts for heat pumps without and with modulation. For both heat pump types the required grid energy decreases with an increasing number of starts.

RH and weighted sum are well-known techniques from optimization literature. We combined both to a optimization strategy which works quite well. We are not aware of other works that have used this approach for dealing with a limited planning horizon.

5.3. Schedule Optimization without Known Energy Production and Consumption Data

In the previous optimizations, we utilized historic energy production and consumption data as input. However, in practice, forecast data are needed for the computation of future schedules. Such forecast data have errors compared to the real future production and consumption data. We first review a simple but yet effective forecast method for energy consumption in single family homes. Then we discuss the effect of forecast errors on optimized schedules and propose the use of a recharge threshold to cope with such errors. We investigate the impact of this recharge threshold to recommend an

Threshold (kWh)	Unsat. demand (%)	Grid energy		On/off cycles	
		Abs. (kWh)	Rel. (%)	Abs.	Rel. (%)
11.233	98	5064	13.1	326	36.4
11.465	58	5078	13.4	334	39.7
11.93	36	5175	15.6	352	47.3
13.325	6	5486	22.5	452	89.1

(a) W/o power modulation.

Threshold (kWh)	Unsat. demand (%)	Grid energy		On/off cycles	
		Abs. (kWh)	Rel. (%)	Abs.	Rel. (%)
11.233	100	4832	13.7	198	32
11.465	62	4810	13.2	202	34.6
11.93	43	4843	13.9	228	52
13.325	5	5018	18.1	296	97.3

(b) W/ power modulation.

Table 6: Required grid energy, on/off cycles, and resulting overall unsatisfied DHW demand for optimized heat pump schedules with forecasts and recharge threshold.

appropriate value. Finally, we quantify the penalty of using forecast data instead of known data.

5.3.1. Forecast Method: 1-Day-Back

Demand for electrical energy, DHW, and space heating are difficult to forecast for single family homes. Especially electrical energy and DHW show almost unpredictable peaks. For simplicity reasons, we just take time series of the previous day as forecast for the succeeding day, which is called 1-day-back method. In [13] we have compared 1-day-back with other, more complex approaches for the single family home in this study, and 1-day-back performed remarkably well for the demand prediction of space heating and DHW as well as PV energy production. Therefore, we use 1-day-back to predict the time series for energy production, energy consumption (space heating, DHW, and other electrical demand), and for outdoor temperature

for the next three days. That means, the respective time series of the previous day are taken as forecast time series for the succeeding three days.

5.3.2. Impact of Forecast Errors

We discuss effects of forecast errors. The real PV energy production may differ from forecast values so that more or less energy is needed from the grid than expected by the schedule. The COP of the heat pump depends on the temperature. Therefore, deviations of the real temperature from the forecast temperature lead to generation of more or less thermal energy by a heat pump than expected by the schedule. If less energy is produced than forecasted, the heat or DHW storage may not be charged to the planned level when operating the heat pump based on an optimized schedule. If more energy is produced than forecasted, the space heating or DHW storage may be charged earlier than planned. If less energy is consumed than forecasted, more energy remains in the space heating or DHW storage than planned. If more energy is consumed than forecasted, the space heating or DHW storage may deplete earlier than planned.

5.3.3. Control Adaption to Cope with Forecast Errors

Forecast errors cause the discussed deviations from the schedule. We adapt the system control to cope with them.

We suggest recharge thresholds for the space heating and DHW storage so that the storages are recharged when their SOC falls below the respective threshold. The recharge threshold can be considered in the optimization by setting H_{min} and W_{min} in Equations (10) and (11) to the recharge thresholds of the space heating and DHW storage instead to their minimum values.

On the one hand, the recharge threshold mechanism reduces the likelihood that space heating or DHW demand meets a fully discharged space heating or DHW storage. On the other hand, this mechanism reduces the flexibility and optimization potential as the SOC of a storage normally ranges between its recharge threshold and maximum state of charge. Thus, less energy can be charged by a single charging process compared to a system without a recharge threshold so that probably more on/off cycles are needed. Thus, the recharge thresholds need to be set carefully.

For the evaluation, we keep a 15-minutes granularity for compatibility with available historic data and the optimization approach. That means, the control algorithm checks the SOC of the space heating and DHW storage at the end of each quarter-hour and triggers appropriate actions. An adaptation

to instant reaction is straightforward. In the following we explain the actions triggered by the control algorithm at the end of each quarter-hour.

If the SOC of a storage exceeds the storage capacity, the SOC is set to the respective maximum and the consumed grid energy is corrected. Then, a new RH optimization is triggered based on an unchanged heat pump state (on/off) and current storage SOCs, and the resulting schedule is carried out for the next quarter-hour.

When the SOC of the DHW storage falls below its minimum allowed value, the difference is recorded as unsatisfied demand. Moreover, the DHW storage is recharged by the heat pump at maximum power and the immersion heater for the next quarter-hour. If the space heating storage undershoots it minimum, the space heating storage is recharged by the heat pump at maximum power. If both storages undershoot their minimum, the DHW storage is recharged by the immersion heater and the space heating storage is recharged by the heat pump at maximum power. However, the space heating storage undershoots its minimum in none of the evaluations.

When the SOC of a storage falls below its recharge threshold but remains above the allowed minimum, that storage is recharged by the heat pump at maximum power which will possibly just be started for that purpose. If the thresholds for both the DHW storage and the space heating storage are undershot, the heat pump recharges the space heating storage at maximum power and the immersion heater recharges the DHW storage.

If the SOC for both storages are between the respective recharge threshold and maximum value, and if the system was operated according to an optimized schedule which does not yet be reoptimized, then this schedule is further carried out in the next quarter-hour. Otherwise, the schedule is reoptimized based on an unchanged heat pump state (on/off) and current storage SOC, and carried out in the next quarter-hour.

5.3.4. Recommendation for the Recharge Threshold

Recharge thresholds exist for space heating and DHW storage and are defined relative to energy of an entirely discharged storage, i.e., 40 kWh and 11 kWh, respectively (cf. Section 3.1).

We choose a recharge threshold of 3.5 kWh for the space heating storage as there is no time slot with more demand in the entire data set. With that parametrization, heat demands can always be satisfied. Moreover, this threshold is relatively small compared to storage size so that the storage remains effective.

DHW behaves differently. The DHW demand of some time slots exceeds the capacity of the DHW storage. To find an appropriate recharge threshold, we evaluate its impact on purchased grid energy, on/off cycles, and unsatisfied demand. Tables 6(a) and 6(b) compile the results for different recharge thresholds. The investigated thresholds correspond to 5%, 10%, 20%, and 50% of the DHW storage capacity. For recharge thresholds of 5%, 10% and 20%, the unsatisfied demand is unacceptably large compared to a yearly DHW demand of 2063 kWh. Therefore, 50% should be chosen, which corresponds to 2.325 kWh. However, larger thresholds reduce the effective DHW storage capacity and lead to more DHW recharges, which can be well observed in Tables 6(a) and 6(b) by the number of increased on/off cycles. The reduced effective DHW storage capacity also decreases the scheduling flexibility, i.e., recharging the DHW storage can wait less often until sufficient PV energy is available.

5.3.5. The Penalty of Using Forecasts

Only in theory future energy production and generation can be known so that no recharge thresholds are needed. This leads to low purchased grid energy and on/off cycles as compiled in Table 5. With forecast data and recharge threshold, between 13.1% and 22.5% more grid energy is needed and between 32% and 97.3% more on/off cycles are required. This is a substantial penalty which shows that this type of optimization problem cannot be treated without considering forecast data.

5.4. Impact of DHW Storage Capacity and PV Power

To avoid substantial unsatisfied demand, a large recharge threshold is needed, which diminishes scheduling flexibility and leads to more purchased grid energy. There are two obvious countermeasures: larger DHW storage capacity or more PV power. We investigate their impact on unsatisfied demand, purchased grid energy, and on/off cycles while keeping the recharge threshold at 2.325 kWh. Tables 7(a) and 7(b) compile the results for default and double DHW storage capacity and PV power.

Double DHW storage decreases unsatisfied demand to almost zero while double PV power has only little positive impact on unsatisfied demand. Double DHW storage capacity saves 546 and 162 kWh grid energy for heat pumps without and with modulation, respectively. The savings through double PV power is 202 and 133 kWh. Thus, both measures reduce the purchased grid energy by some moderate amount. However, double DHW storage capacity

PV power	Default DHW storage capacity			Double DHW storage capacity		
	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles
Default	6	5486	452	0	4940	270
Double	4.3	5282	451	0	4175	275

(a) W/o power modulation.

PV power	Default DHW storage capacity			Double DHW storage capacity		
	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles
Default	**5**	**5018**	**296**	**0.5**	**4724**	**174**
Double	5	4885	321	0	4213	186

(b) W/ power modulation.

Table 7: Unsatisfied overall DHW demand, required grid energy, and on/off cycles for heat pumps running on optimized schedules; the recharge threshold is 2.325 kWh, default and double PV storage capacity and PV power are investigated.

reduces the number of on/off cycles by 40% and 41% for heat pumps without and with modulation. In contrast, double PV power hardly decreases on/off cycles or even increases them. Hence, double PV power is less effective than double DHW storage capacity and more expensive. As double DHW storage capacity is an effective means to reduce unsatisfied demand, purchased grid energy, and on/off cycles, we use a 200 l DHW storage with 9.3 kWh thermal capacity together with a recharge threshold of 2.325 kWh in the remainder of this work.

5.5. Comparison with a Simple Control Strategy

A simple control strategy also requires recharge thresholds for space heating and DHW. We use them as proposed. The recharge thresholds are the only triggers to start the heat pump. The heat pump is always operated at full rate so that advantages of heat pumps with power modulation cannot be leveraged. Whenever the energy in the DHW storages falls short of the corresponding recharge threshold, the DHW storage is recharged, otherwise

the space heating storage is recharged, i.e., recharging DHW has priority. When both storages are full, the heat pump is switched off.

Tables 8(a) and 8(b) compile results for different DHW recharge thresholds and for default and double DHW storage sizes. We observe that the unsatisfied demand is larger than for optimized schedules (cf. Tables 7(a) and 7(b)) for corresponding recharge thresholds. Although the simple control strategy is unable to leverage advantages of heat pumps with power modulation, we still compare their performance to the one of heat pumps with power modulation running on optimized schedules. Table 8(a) shows that for the default DHW storage capacity, the simple control strategy requires 1323 kWh more grid energy and 296 more on/off cycles while the unsatisfied demand is 15 kWh higher. For the double DHW storage capacity in Table 8(b), the simple control strategy requires 1107 kWh more grid energy and 111 more on/off cycles. Tables 8(a) and 8(b) also states the relative increase of purchased grid energy and on/off cycles for the simple control strategy compared to the optimized control strategy.

Given the fact that at least 2340 kWh electrical demand for household must be supplied by the grid due to unavailability of PV energy, optimized schedules are able to reduce the remaining demand from 3491 kWh to 2384 kWh (-31.7%) for double DHW storage capacity, which is essentially the benefit of the presented optimized control.

5.6. Analysis of System Behavior

In the following we analyze the system behavior. We study how heat pumps without and with power modulation are utilized with optimized schedules. We show how electrical energy is consumed from and supplied to the grid over the year. We investigate when DHW is generated by the heat pump and by the immersion heater, respectively. Finally, we study the use of an immersion heater with less power as an alternative. The system under study uses schedules optimized with RH and weighted-sum optimization, the optimization takes forecast data as input. A recharge threshold of 2.325 kWh is used to minimize unsatisfied DHW demand, and the double DHW storage capacity (200 l, 9.3 kWh) is utilized for better efficiency.

5.6.1. Modulation Behaviour

The previous evaluations revealed that heat pumps with powe modulation offer the potential for less purchased grid energy and fewer on/off cycles

Recharge threshold (%)	Unsat. demand (kWh)	Grid energy		On/off cycles	
		Res. (kWh)	Inc. (%)	Res.	Inc. (%)
11.233	134	6149	21.4	391	20.0
11.465	109	6154	21.2	403	20.7
11.93	60	6181	19.4	425	20.7
13.325	**21**	**6341**	**15.6**	**592**	**30.1**

(a) Default DHW storage capacity.

Recharge threshold (%)	Unsat. demand (kWh)	Grid energy		On/off cycles	
		Res. (kWh)	Inc. (%)	Res.	Inc. (%)
11.233	37	5820	23.7	240	49.1
11.465	32	5818	23.1	242	46.7
11.93	17	5920	25.5	251	53.0
13.325	**0.5**	**5831**	**23.4**	**285**	**64.7**

(b) Double DHW storage capacity.

Table 8: Unsatisfied overall DHW demand, required grid energy, and on/off cycles for a simple control strategy with forecast data for different recharge thresholds and DHW storage capacities. The simple control strategy operates the heat pump always at maximum power. The indicated increase is relative to a heat pump with modulation running on optimized schedules (cf. bold numbers in Tables 7(a) and 7(b)).

than heat pumps without modulation when executing optimized schedules. Therefore, we study their modulation behaviour.

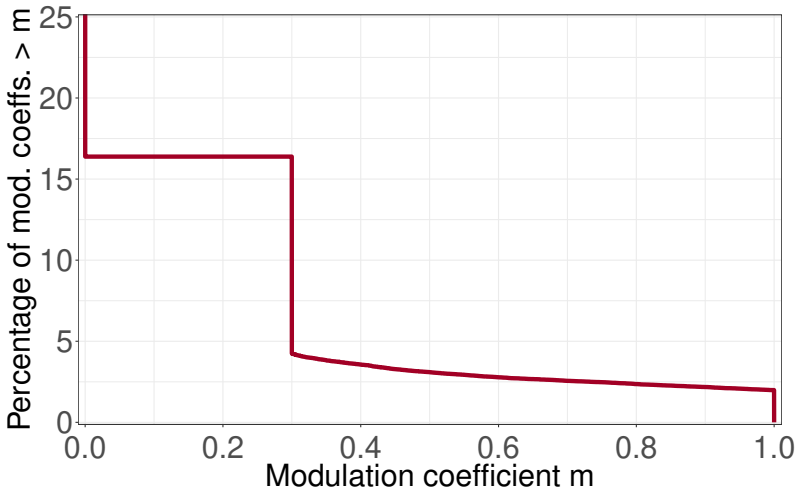


Figure 6: CCDF of the modulation coefficients on a quarter-hous basis for a heat pump w/ power modulation running on optimizes schedules over a year; simple forecasts are used as optimization input, double DHW storage capacity, and a recharge threshold of 2.325 kWh; results for default DHW storage are almost identical.

Figure 6 shows the the complementary cumulative distribution function (CCDF) of modulation coefficients over time. The heat pump is off in 83.8% of all quarter-hours during the year and on in 16.2%. In 11.7% of the time, the heat pump runs with the minimum modulation coefficient of 30% and in 2.0% of the time, the heat pump runs at 100%. The remaining 2.5% of the time, the heat pump works with a modulation coefficient in the range between 30% and 100%.

We explain why low modulation coefficients, which are preferably used by optimized schedules, are beneficial to reduce purchased grid energy and the number of on/off cycles. PV power is limited to about 5 kW in summer and it is mostly lower in other seasons. The maximum electrical power of the heat pump depends on the outer temperature. It is mostly between 2.25 kW and 3.12 kW for generating heat for space heating and between 3.41 kW and 4.66 kW for generating DHW. Thus, the electrical power needed by the heat pump in full operation generally exceeds the supplied PV power. Therefore, operating the heat pump at a low modulation coefficient reduces the purchased grid energy. Another aspect is that the heat pump has longer runtimes at a low modulation coefficient until the limited thermal storages

are charged. Thus, when the thermal demand is high, the heat pump can continuously run without interruptions due to fully charged thermal storages, which saves on/off cycles. This hypothesis will be backed by the analysis of the heat pump's runtime behavior.

5.6.2. Runtime Behavior of the Heat Pump

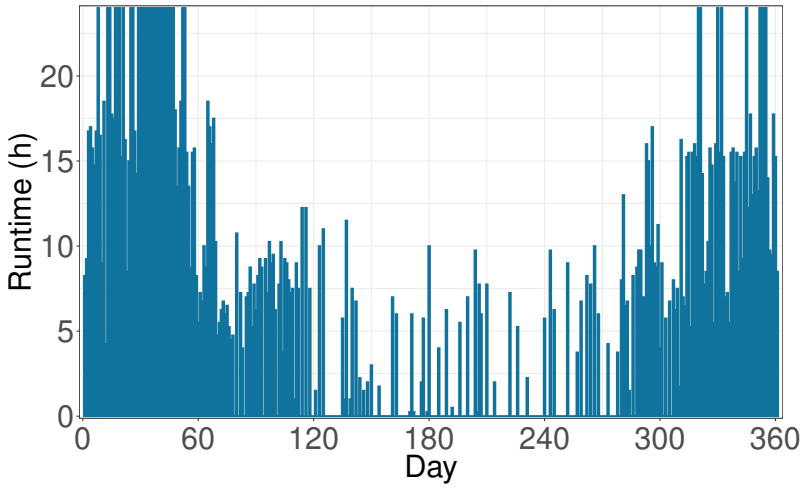
Figures 7(b)–7(a) depict daily times of operation over a year for heat pumps running on optimized schedules. The runtime of heat pumps clearly follows a seasonal pattern as they are longer in winter than in summer as their operation is triggered by the thermal demand. Since the demand for space heating is almost zero in summer, most of the short summer runtimes can be attributed to the recharge of the DHW storage. We observe clear differences in the times of operation for heat pumps without and with power modulation. First, the times of operation for heat pumps with power modulation are significantly longer than those for heat pumps without modulation. In fact, the heat pump with modulation runs for 24 h during many days in winter, which obviously saves on/off cycles. Moreover, the accumulated yearly runtime for the heat pumps is 2838 and 1191 hours, respectively. We point out that the lifetime of a heat pump not only depends on the number of on/off cycles, but also on its overall runtime. This, however, is certainly a technology- and vendor-specific issue and goes beyond the scope of this paper.

5.6.3. Grid Energy Demand and Supply

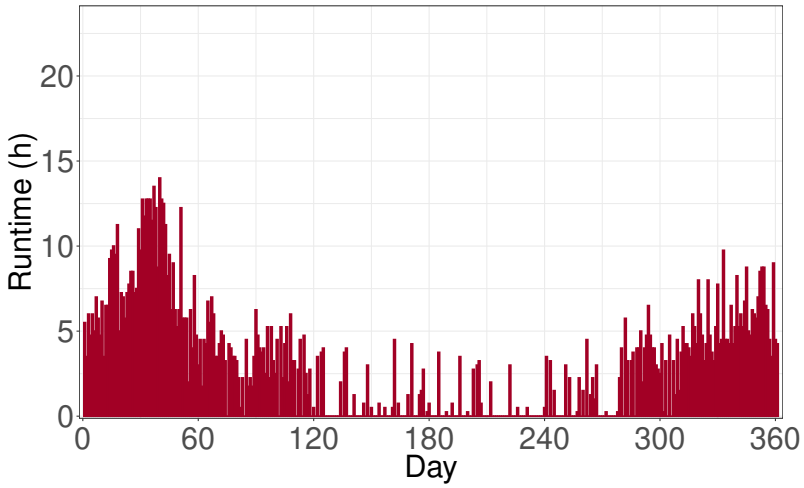
PV is not always sufficiently available when needed and, conversely, cannot always be fully consumed due to lack of demand. Thus, energy is needed from the grid, or can be fed into the grid. Figure 8 shows the residual grid energy demand and supply when a heat pump with power modulation running on optimizes schedules is used (double DHW storage capacity). In winter, energy is mostly demanded from the grid, in summer, energy is mostly supplied to the grid. However, almost everyday, also in summer, some energy is demanded from the grid. An amount of 2340 kWh electrical demand from the household cannot be covered by PV energy.

5.6.4. Use of the Immersion Heater

Figure 9(a) illustrates how DHW is generated by the heat pump or the immersion heater, respectively. At first sight, the frequent usage of the immersion heater may be surprising, especially during summer. There are basically two reasons for the use of the immersion heater. First, it is used if



(a) W/ modulation; 2838 hours accumulated.



(b) W/o modulation; 1191 hours accumulated.

Figure 7: Daily runtimes over a year for heat pumps running on optimized schedules; simple forecasts are used as optimization input, double DHW storage capacity, and a recharge threshold of 2.325 kWh.

the heat pump is not sufficient to charge space heating and DHW storage up to the recharge threshold until the end of an quarter-hour within the op-

timization period. In particular, the heat pump can charge either the space heating or the DHW storage within a quarter-hour. This causes the need for the short usages of the immersion heater in winter. Second, in summer the immersion heater is used instead of the heat pump to keep the number of on/off cycles low.

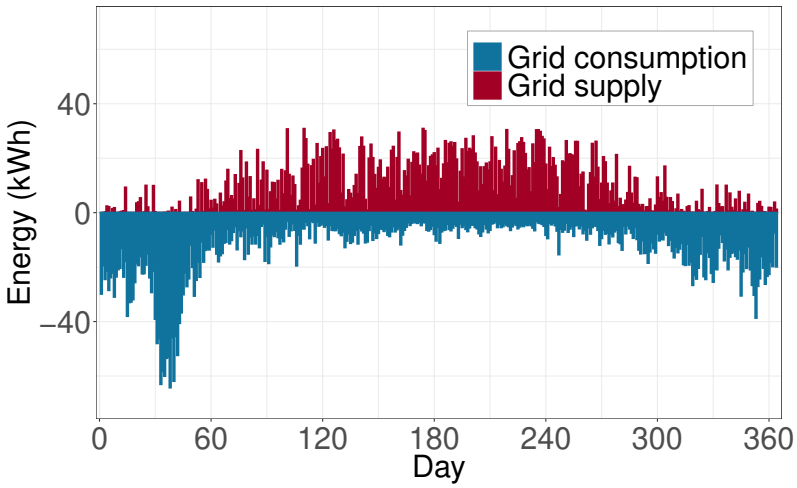
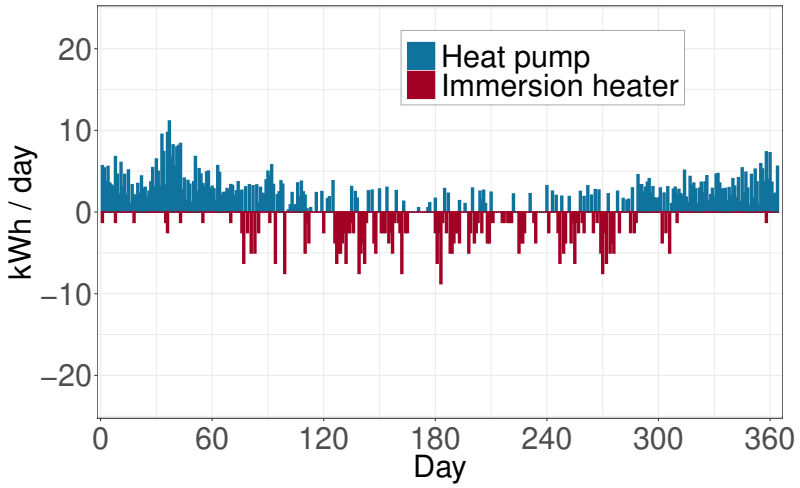
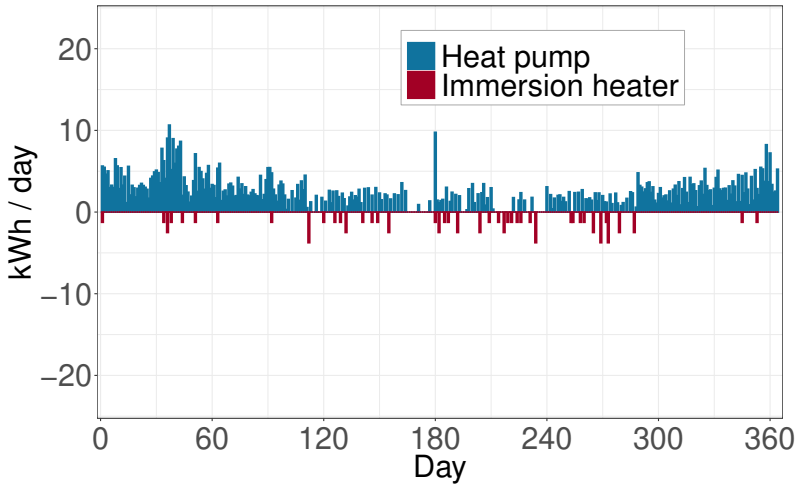


Figure 8: Daily grid energy supply and demand over a year; a heat pump with modulation is used, simple forecasts as optimization input, double DHW storage capacity, and a recharge threshold of 2.325 kWh.

We explain the latter by a closer look at the optimized schedules. The heat pump is used only for generating DHW if this can be combined with generating heat for space heating, which is due to the optimization approach using virtual start costs. For a virtual start cost of $C = 5$ kWh, the heat pump is started for only generating DHW if this can save at least $C = 5$ kWh electrical energy from the grid compared to generating DHW with the immersion heater. Without any PV energy and a COP of 3, at least 7.5 kWh DHW demand is needed to activate the heat pump. If some PV energy is available, the heat pump may be modulated so that no grid energy is needed. However, also the immersion heater requires less grid energy. For instance, when 2.5 kW PV power is available, the immersion heater draws only half of its energy from the grid. Therefore, at least 10 kWh DHW demand are needed to trigger the heat pump to start when no additional heat for space



(a) Virtual start cost $C = 5$ kWh.



(b) Virtual start cost $C = 1$ kWh

Figure 9: Electrical energy used by an immersion heater and a heat pump with modulation to recharge the DHW storage.

heating is needed. This is more than the capacity of the considered double storage and explains that the immersion heater is even used when it consumes almost 9 kWh for DHW in summer.

C (kWh)	Grid energy (kWh)	On/off cycles	Unsat. demand (kWh)	Heater runtime (h)
1	4489	285	0.2	15.5
2	4615	216	0.6	47.5
2.5	4650	200	0.4	51.25
5	4727	173	0.5	74.5

Table 9: Impact of virtual start cost on usage of the immersion heater and other metrics.

We varied the virtual start cost to demonstrate its impact on the operation of the immersion heater. Table 9 shows that lower values lead to fewer operation hours of the immersion heater and also to less required grid energy, but also to more on/off cycles. We illustrate the different cumulated operating hours of the immersion heater when a small virtual start cost of $C = 1$ kWh is used for schedule optimization. Figure 9(b) shows that DHW is generated less frequently by the immersion heater compared to $C = 5$ kWh in Figure 9(a). Instead, the heat pump is used more frequently. Moreover, the immersion heater is utilized only for small recharges of the DHW storage. We remark that the trade-off between reducing grid energy consumption and reducing the number of on/off cycles depends on practical considerations for specific devices and are beyond the scope of this work.

C (kWh)	3 kW immersion heater				5 kW immersion heater			
	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles	Heater runtime (h)	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles	Heater runtime (h)
1	0.4	4612	374	152	0.4	4607	388	40
2	0.9	4674	322	174	0.7	4716	331	79
5	0.6	4764	286	201	0	4940	270	128

(a) W/o power modulation.

C (kWh)	3 kW immersion heater				5 kW immersion heater			
	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles	Heater runtime (h)	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles	Heater runtime (h)
1	0.1	4581	255	148	0.2	4489	285	19
2	0.4	4647	207	152	0.6	4615	216	40
5	0.3	4740	169	183	0.5	4724	174	70

(b) W/ power modulation.

Table 10: Impact of start cost on usage of the immersion heater and other metrics using forecast data as optimization input.

C (kWh)	3 kW immersion heater				5 kW immersion heater			
	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles	Heater runtime (h)	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles	Heater runtime (h)
1	0	4012	278	160	0	4033	280	25
2	0	4031	264	173	0	4087	252	54
5	0	4087	248	183	0	4154	237	66

(a) W/o power modulation.

C (kWh)	3 kW immersion heater				5 kW immersion heater			
	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles	Heater runtime (h)	Unsat. demand (kWh)	Grid energy (kWh)	On/off cycles	Heater runtime (h)
1	0	3891	214	157	0	3905	223	16
2	0	3939	176	166	0	3963	185	42
5	0	4047	138	160	0	4108	136	63

(b) W/ power modulation.

Table 11: Impact of start cost on usage of the immersion heater and other metrics using historical data as optimization input.

5.6.5. Impact of an Immersion Heater with Less Power

A heat pump can be modulated such that its effective power is below the predicted PV power and no grid energy is needed in the best case. In contrast, the immersion heater cannot be modulated and draws 5 kW during operation. This power can be partially provided by PV, but the residual power is drawn from the grid. In the same situation, an immersion heater with less power causes less residual energy.

We carried out experiments with a 3 kW immersion heater and compiled their results in Tables 10(a) and 10(b). We observe that the immersion heater with lower power reduces the required grid energy for heat pumps without modulation and for heat pumps with modulation and the default storage capacity. However, we do not see the same effect for a heat pump with modulation and the double storage. This finding is consistent for different virtual start cost.

We compare the results to those of corresponding experiments with historical data as optimization input, i.e., full knowledge instead of forecast data. Tables 11(a) and 11(b) reveal that the difference in grid energy between the 5 kW immersion heater and the 3 kW immersion heater is larger for historical input data than for forecast input data. Thus, forecast input data diminish the advantage of the immersion heater with low power. We also observe that this difference is larger for heat pumps with power modu-

lation which leads to longer operation times for the immersion heater. We suspect that wrong forecast input data diminish the reduction in grid energy, which is more significant for immersion heaters with low power as they have longer operation times. This advantage of reduced grid energy for immersion heaters with low power even vanishes completely for double storage capacity.

This finding shows that schedules for heat pumps can generally be well optimized with coarse forecast data, but some variants are more susceptible to forecast errors than others.

6. Conclusion

We proposed an integer linear program (ILP) to compute schedules for a heat pump that is operated partly by PV energy and partly by electricity purchased from the grid. Objectives are minimization of both, purchased grid energy and the number of the heat pump's on/off cycles as the latter extends the heat pump's lifetime. The heat pump provides domestic hot water (DHW) and for space heating which can be stored in a tank or in the floor heating mass, respectively. Several challenges had to be tackled. Incremental optimization over time is needed as forecasts for energy production and consumption are feasible only for a few days, which has been solved by a rolling horizon approach with an optimization horizon of three days. This significantly reduces the size of the optimization problem so that it can be solved within 2 minutes on a Raspberry Pi. The joint minimization of purchased grid energy and on/off cycles represents a multi-objective problem which has been solved by weighted sum optimization with appropriate virtual start costs. When using real forecasts instead of known data, the actual energy production and consumption may deviate from the predicted values such that the energy in the DHW storage and floor heating may deviate from its predicted evolution. This may lead to space heating/DHW storage depletion and unsatisfied space heating/DHW demand. To reduce the likelihood for such events, we proposed that the heat pump is started irrespectively of the current schedule when the energy in the space heating/DHW storage falls below a threshold, which also triggers a re-computation of the heat pump's schedule. A comparison with simple heat pump operation showed that 1000 kWh (20%) purchased electrical energy can be saved in the considered real-world example and that 40% of the on/off cycles could be saved. With optimized schedules, heat pumps with power modulation mostly run with a low modulation coefficient so that they run over longer periods than

heat pumps without power modulation and lead to fewer on/off cycles. Moreover, a sufficiently large DHW storage is recommendable so that the number of on/off cycles can be kept low and unsatisfied demand can be avoided. As the utilized forecasts do not need external data and the computation needs for the optimization are low, the proposed method can be well applied in practice. In future research, its applicability should be tested in practice and for houses with various properties.

Finally, we remark that this work presented evaluations based on a theoretical model. COPs were assumed to be independent of the modulation factor. We did not quantify the benefit of fewer on/off cycles on maintenance intervals and maintenance costs, which would be interesting. Further, heat pumps operated with a lower modulation factor have longer running times, which may adversely affect these metrics. Those are practical and economical considerations which are beyond the scope of this work and should be investigated in future studies.

References

[1] K. D. Le, J. T. Day, Rolling Horizon Method: A New Optimization Technique for Generation Expansion Studies, IEEE Transactions on Power Apparatus and Systems PAS-101 (9) (1982) 3112–3116.

[2] M. Akmal, B. Fox, Modelling and Simulation of Underfloor Heating System Supplied from Heat Pump, in: International Conference on Computer Modelling and Simulation, 2016, pp. 246–251.

[3] A. Allouhi, et al., Simulation of a Thermoelectric Heating System for Small-size Office Buildings in Cold Climates, in: International Renewable and Sustainable Energy Conference, 2015, pp. 1–6.

[4] T. Péan, et al., Experimental Testing of Variable Speed Heat Pump Control Strategies for Enhancing Energy Flexibility in Buildings, IEEE access 7 (2019) 37071–37087.

[5] M. Mastouri, N. Bouguila, A Methodology for Thermal Modelling and Predictive Control for Building Heating Systems, in: International Conference on Sciences and Techniques of Automatic Control and Computer Engineering, 2017, pp. 568–573.

[6] L. Song, et al., Analysis of Micro-grid Integration with PV, Energy Storage and Ground-source Heat Pump Based on DeST Simulation, in: Conference on Energy Internet and Energy System Integration, 2017, pp. 1–4.

[7] B. Verbruggen, J. Driesen, Grid Impact Indicators for Active Building Simulations, Transactions on Sustainable Energy 6 (1) (2014) 43–50.

[8] J. Rimbala, et al., Assessment of Energy Consumption in the Residential Building with a Heat Pump, in: International Scientific Conference on Electric Power Engineering, 2019, pp. 1–5.

[9] S. L. Tangwe, et al., Prediction of Coefficient of Performance and Simulation Design of an Air-source Heat Pump Water Heater, Journal of Engineering, Design and Technology (2017).

[10] M. Loesch, et al., Demand Side Management in Smart Buildings by Intelligent Scheduling of Heat Pumps, in: International Conference on Intelligent Energy and Power Systems, 2014, pp. 1–6.

[11] M. Hall, A. Geissler, Einfluss der Wärmespeicherfähigkeit auf die energetische Flexibilität von Gebäuden, Bauphysik 37 (2) (2015) 115–123.

[12] L. Zadeh, Optimality and Non-scalar-valued Performance Criteria, IEEE Transactions on Automatic Control 8 (1) (1963) 59–60.

[13] T. Stüber, R. Hogl, B. Thomas, M. Menth, Comparison of Forecasting Methods for Energy Demands in Single Family Homes, in: ETG Congress 2021, 2021, pp. 1–5.

2.2 Efficient Robust Schedules (ERS) for Time-Sensitive Networking

Efficient Robust Schedules (ERS) Time-Aware Shaping for Time-Sensitive Networking

Thomas Stüber, Lukas Osswald, Michael Menth

Chair of Communication Networks, University of Tuebingen, Germany
{thomas.stueber, lukas.osswald, menth}@uni-tuebingen.de

***Abstract*—Time-Sensitive Networking (TSN) extends Ethernet bridging with features for deterministic transmission. Periodic streams may be scheduled such that their frames hardly interfere in bridges. Additionally, the Time-Aware Shaper (TAS) can keep egress ports free from other traffic when scheduled traffic arrives. TAS scheduling determines transmission starts of scheduled streams at end stations and configures the TAS in bridges. Most TAS scheduling algorithms disregard jitter and synchronization errors at end stations and bridges, race conditions from simultaneously arriving frames with same egress ports, and hardware-based configuration limits of the TAS. We call the resulting schedules tight schedules (TS). However, all these challenges apply to real hardware bridges. Therefore, we present an algorithm using event times with uncertainty to compute efficient robust schedules (ERS) that respect these constraints. We also propose a repair for existing scheduling approaches and call their output naïve robust schedules (NRS). We evaluate and compare their bandwidth usage and stream admission with those of TS. ERS are more efficient than NRS, and the performance gap between ERS and TS quantifies the price for robust schedules. Moreover, the presented algorithm for ERS computes significantly faster than four well-known methods for TS, and it can solve larger problem instances.**

***Index Terms*—Time-sensitive networking (TSN), Time-aware shaper (TAS), Scheduling, Optimization, Real-Time communication**

I. Introduction

Time-Sensitive Networking (TSN) is a set of IEEE standards that extend Ethernet bridging (IEEE 802.1Q [1]) to provide deterministic data transmission using eight priority queues. Time-triggered (TT) traffic consists of high-priority periodic streams with very low end-to-end delay requirements, also called streams. Amongst other Quality of Service (QoS) features, TSN supports scheduling TT streams so that their frames hardly interfere at any forwarding node and experience only little queuing delay, if any. To ensure that links are not occupied by other traffic when scheduled traffic should be sent, IEEE 802.1Qbv [2] introduces an enhancement for scheduled traffic, commonly denoted as Time-Aware Shaper (TAS). It periodically allows and prevents queues to send frames to their egress port. The periodic behavior is defined through a Gate Control List (GCL) with a limited number of entries. Each entry can open or close the gates of the queues. Thereby, exclusive time slots can be provided for TT traffic so that it cannot be delayed by lower-priority traffic.

Scheduling TT traffic comprises the assignment of periodic sending times to TT streams at end stations and the configuration of the GCL entries for the TAS in forwarding nodes. The latter is also called GCL synthesis. There is a significant body of literature on scheduling TT traffic in TSN [3]. However, almost all works make idealized assumptions which do not hold for hardware bridges. They assume exact sending times at end stations, constant processing delays in forwarding nodes, perfect time synchronization among bridges and end stations, no race conditions for almost simultaneously arriving frames with the same egress port, and an unlimited number of GCL entries. We have experimented with TSN hardware and have experienced that such schedules may fail in practice. When a frame arrives only little earlier or later than scheduled, it may be sent during an earlier or later time slice than scheduled. As a consequence, frames may miss their deadlines and the deviation of a single frame from the schedule may delay other frames, resulting in even more problems. The miss of deadlines is not tolerable for TT traffic as it may impose safety risks for hard real-time use cases, e.g., the transmission of critical sensor data in in-vehicle networks.

In this work, we argue that most algorithms for TAS scheduling from the literature disregard the mentioned challenges. We call them tight schedules (TS) as they do not leave sufficient space between frames within a schedule to compensate for hardware jitter. Some algorithms prevent race conditions for frames with joint egress ports and a few works also propose an extension against synchronization errors [4][5][6]. We show that this extension is insufficient and fix it in a simple but inefficient way, leading to naïve, robust schedules (NRS). We further propose an ILP-based scheduling algorithm taking all the mentioned challenges into account. It considers arrival and transmission times of frames as event times with uncertainty. Frames are scheduled such that they cannot interfere with each other and that they meet their deadlines as long as jitter for initial transmission and for processing, as well as synchronization errors, remain within assumed bounds. This method results in efficient, robust schedules (ERS). The ILP maximizes the available resources for non-scheduled traffic as an objective function, and is designed in a way that TS and NRS can be computed with a subset of its constraints.

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under support code 16KIS1161 (Collaborative Project KITOS). The authors alone are responsible for the content of the paper.

Subsequently, we evaluate and compare the different schedule types (TS, NRS, ERS) with regard to bandwidth usage and number of admissible streams. As expected, TS are more efficient than ERS, quantifying the price of robustness which is prerequisite for application in practice. Furthermore, ERS are more efficient than NRS, showing that modelling jitter using event times with uncertainty matters. To assess the scalability of the ERS approach, we compare its runtime with the runtimes of well-known scheduling algorithms from the literature. These algorithms yield only TS or NRS-like schedules. It turns out that the ERS approach computes significantly faster and is able to solve larger problem instances.

This work does not raise any ethical issues. The remainder of the paper is structured as follows. We introduce fundamentals of TSN and the TAS in Section II. Section III states the problem considered in this work. Section IV gives an overview of related work for per-stream scheduling with the TAS. We demonstrate that the existing approach to cope with synchronization errors is flawed and propose a repair in Section V. We explain measures to counteract jitter in processing delays, synchronization errors, and race conditions in Section VI. Section VII sketches the structure of the ILP-based scheduling algorithms for ERS, NRS, and TS while details about the ILP are given in the appendix. In Section VIII the schedule types ERS, NRS, and TS are evaluated and compared with regard to bandwidth usage and number of admissible streams, and the runtime for ERS computation is compared to well-known methods from the literature. Finally, we conclude the paper in Section IX.

II. Time-Sensitive Networking (TSN)

Time-Sensitive Networking (TSN) is a set of standards for real-time communication over Ethernet networks. It is organized in standards for time synchronization, scheduling and traffic shaping, as well as resource and network management. Devices such as bridges and end stations need a common understanding of time to enable traffic scheduling. Therefore, every device is equipped with a clock. These clocks are synchronized with a protocol defined in IEEE 802.1AS [7]. It allows sub-microsecond precision for reasonably sized network topologies. Periodic real-time traffic is denoted as time-triggered (TT) traffic. To enable the scheduling of TT streams with hard real-time constraints, TSN defines a set of possible traffic shaper mechanisms.

TSN introduces an enhancement for scheduled traffic in IEEE 802.1Qbv [2]. The features of this enhancement can be used to implement the Time-Aware Shaper (TAS). Figure 1 depicts the components of the TAS in an egress port. Every egress port is equipped with eight egress queues. These queues are First-in-First-out (FIFO), i.e., there is no reordering and only the longest waiting frames can be dispatched. Frames are assigned to queues based on the Priority Code Point (PCP) field of the VLAN tag in their headers. The TAS periodically opens and closes so-called transmission gates of these queues. Its objective is to protect scheduled streams from other traffic, e.g., from best-effort traffic. The behavior of the TAS is

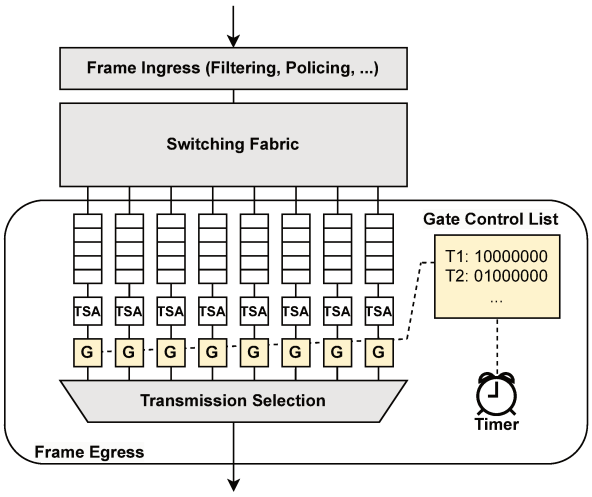


Fig. 1: Path of a frame through a bridge implementing the TAS.

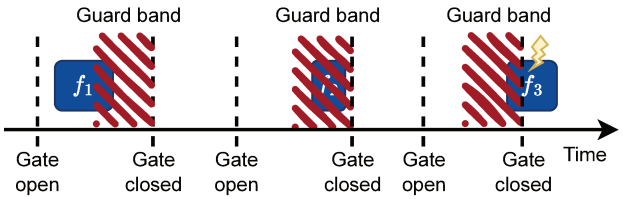


Fig. 2: Transmission during a guard band is allowed if the transmission finishes before the gate closes.

controlled by a so-called Gate Control List (GCL). A GCL entry contains a time interval and a bit-vector indicating which gates are opened and closed during this interval. Within that interval, only queues with an open gate can send traffic to the egress port. The GCL entries are executed periodically, and bridges support only a limited number of GCL entries. Every queue is guarded by a transmission selection algorithm (TSA) which signals whether a frame can be dispatched. Among the queues with such a positive indication, a transmission selection mechanism selects the queue from which the next frame is dispatched. We assume that strict priority is implemented, i.e., transmission selection chooses among all queues with a positive indication and an open gate the one with the highest priority.

Traffic scheduling implies synchronizing the clocks of all network devices, choosing appropriate transmission offsets for scheduled streams at end stations, and coordinating the GCLs of the bridges. Scheduled traffic hardly interferes with other frames so that it experiences only very little queuing delay. As a result, arrival deadlines of individual frames can be guaranteed at their respective destinations. Schedules of transmission offsets and GCLs are periodic, i.e., they can be repeated an indefinite number of times.

A frame transmission is only allowed to start if it will be finished before the next gate closing of the respective egress

queue. Bridges send frames only if this condition is met. This results in a time interval not available for arbitrary sized frame transmissions before a gate closing event. This time interval is commonly denoted as *guard band*. We emphasize that guard bands are implicit in TSN, i.e., the gate is not explicitly closed during a guard band and frame transmissions are not forbidden. However, frames may be held back due to their size during a guard band and hence bandwidth may be wasted due to blocking. Figure 2 depicts three scenarios for frame transmissions during a guard band. Frames f_1 and f_2 are transmitted during a guard band, but transmission finishes before the gate is closed. However, the transmission of f_3 starts too close before the gate is closed. This will not happen in practice, as a bridge detects this conflict and hold back f_3 until the gate is opened for a sufficient amount of time. As queues are FIFO, f_3 may also block the transmission of smaller frames that could be transmitted before the gate closes. Therefore, bandwidth may be wasted within guard bands.

The length of a guard band depends on the maximum frame size and the transmission rate of the egress port. We assume standard Ethernet frames with a maximum size of $1542\,B$, including preamble and inter-frame gap. Therefore, the length of a guard band is $\frac{1542\,B}{r_{\text{trans}}}$ where r_{trans} is the transmission rate of the respective egress port.

III. Problem Statement

We first identify sources of non-determinism and then formulate a problem statement, i.e., the computation of a TSN schedule that is robust against the mentioned non-determinism.

A. Missing Frames

TT streams are periodic streams. If a frame of such a stream is suppressed by a source or just dropped for some reason, the frame is missing from the perspective of the calculated schedule. If this frame normally delays other frames within an egress queue according to the schedule, these other frames will be sent earlier when the frame is missing. This may have detrimental effects on their downstream links and may cause other frames to miss their deadlines. This source of non-determinism has been understood years ago, and Craciunas proposed frame isolation [4] as a countermeasure. That is, at most a single frame or only frames of a single stream may wait at a time in an egress queue.

B. Processing Delay

In contrast to common assumptions in the TSN scheduling literature, the processing delay of a hardware bridge is not constant due to physical effects. Therefore, the arrival time of a frame at an egress queue is subject to jitter. The jitter may accumulate over multiple hops or may lead to a frame being transmitted in the wrong time slice. This may result in a stream missing its deadline.

C. Clock Synchronization Error

Another assumption is that the internal clocks of bridges and end stations run in perfectly synchronized manner. This is not the case in real networks. Even if a frame is sent at the exact time based on the sending node's clock, it may arrive earlier or later than scheduled at the receiving node based on the receiving node's clock. This may result in similar, but not exactly the same problems, as for the jitter of processing delays.

We formalize the effect of synchronization errors. Assume a frame f is sent from device A to device B at reference time t. As A and B have own clocks that may not be perfectly synchronized, we indicate their local time by $t_A(t)$ and $t_B(t)$. All devices execute the schedule based on their own clock. That means, if $t_A(t) < t_B(t)$ holds, f arrives later than scheduled from the perspective of B. The opposite is true if $t_A(t) > t_B(t)$. Then f arrives earlier than expected from B's perspective when it was sent in time from A's perspective.

D. Race Conditions

Race conditions occur when two frames simultaneously arrive at different ingress ports but are forwarded by the same egress port. Then the order of frame transmissions may change non-deterministically. As a result, a delayed frame may miss its deadline.

E. Limited Number of Supported GCL Entries

Finally, real hardware bridges have only a limited number of GCL entries per egress port as they constitute physical resources. Typical numbers are not clear yet. If a schedule utilizes more GCL entries than supported for an egress port, the schedule cannot be deployed. This seems trivial, but the majority of scheduling algorithms plans with an unlimited number of schedule entries and cannot take a limit as input.

F. Computation of Robust Schedules for TSN

The problem statement can be described as follows. Given a network topology and a set of time-triggered periodic streams with deadlines. Compute transmission offsets of all frames at their source nodes and GCLs for all egress ports of bridges such that every frame arrives at all of its destinations before its deadline expires. The schedule must be robust within given bounds against missing frames, jitter in processing delays, unsynchronized clocks, and race conditions of frame arrivals, i.e., all streams meet their deadlines regardless of these non-deterministic effects. Additionally, the schedule must only use a limited number of GCL entries per egress port to be deployable to hardware bridges. A valid schedule must be periodic itself, i.e., it is repeated an indefinite number of times.

The period of a schedule is the least common multiple of the periods of all streams. This duration is commonly denoted as *hyperperiod* H. Various constraints restrict the set of valid schedules. New constraints to achieve robustness are derived in Section VI and Section VII-D2, others are well-known from the literature. The resulting ILP model is discussed in Section VII.

IV. Related Work

We give an overview of related research works that address some challenges mentioned in Section III. First, we review scheduling algorithms that do not consider robustness against non-determinism. Then, we discuss scheduling approaches featuring robustness against time synchronization errors. Finally, we summarize the research gap to the related work.

A. Scheduling without Robustness Features

Early research work by Dürr et al. [8] pursues zero-queuing, i.e., it schedules streams such that they do not wait at all at gates. However, the gates are still needed to protect scheduled traffic against non-scheduled traffic. The authors identified the problem of guard bands which consume bandwidth. To cope with this problem, they proposed a schedule compression algorithm as postprocessing, which reduces the number of guard bands. The algorithm cannot limit the number of used GCL entries to an upper bound, it does not take non-deterministic effects into account, and it does not consider guard bands or maximize bandwidth for other traffic during optimization.

The algorithm from [8] is extended by Hellmanns et al. [9] for the special case of networks with multi-layer ring topologies. The authors state that the number of GCL entries needed is significantly reduced by their 2-step scheduling approach, but it cannot limit their number to a given value.

Dos Santos et al. [10] present an extensive SMT (Satisfiability Modulo Theories) model for scheduling which includes GCL synthesis. It is used for scheduling in the well-known simulation framework OMNeT++ [11]. However, they do not include considerations for non-determinism and only allow a single gate closing per egress port within a hyperperiod.

B. Scheduling with Robustness Features

Craciunas et al. [4] introduced frame and flow isolation constraints (cf. Section III-A) for robustness against missing frames in periodic streams. They showed that that isolation constraints reduce the number of admissible streams. Several research works adopted these isolation constraints, e.g., [6], [12], [13], [5], [14], [15] [16], [17], and [18], only to name a few. Craciunas et al. [4] allow multiple queues for scheduled traffic per egress port and assign every stream to an egress queue per egress port. The authors also identified the problem of clock differences and proposed a solution. However, their approach is flawed which has not been discussed in the literature so far. We revisit their method in Section V and suggest how to cope with clock synchronization errors in a different way. The mentioned work neither limits the number of GCL entries used by a schedule, nor does it consider jitter in processing delays.

In [6] Craciunas et al. extend their model from [4] to compute schedules robust against clock drift, i.e., clocks running with different speeds. The problem was solved by increasing the parameter for the maximum clock synchronization error from [4] and also suffers from the problems discussed in Section V.

Oliver et al. [5] present an SMT model with transmission windows for egress ports. Transmission windows correspond to time slices and the GCL can be derived from them. Instead of scheduling transmission offsets of streams, streams are assigned to these transmission windows. As the number of transmission windows is predefined and fixed, the number of GCL entries needed to deploy a schedule is also fixed. This work handles clock synchronization errors essentially in the same way as [4] and therefore suffers from the same problem. A major obstacle for the use of this method in practice is its computation complexity. The model grows with the number of available GCL entries. The authors report solving times of more than 40 h for problem instances with 50 streams, 10 bridges, and 64 GCL entries.

Jin et al. [19] propose an SMT model to cope with a limited number of GCL entries per egress port. They extend transmission delays by maximum synchronization errors, but do not consider processing jitter. Their model discretizes time and grows with the number of time units per schedule, leading to poor scalability (cf. Section VIII-D).

C. Research Gap

Robustness against missing frames is currently adopted in some works on TAS scheduling, e.g., [4], [5], and [19]. Only a few authors consider a limited number of GCL entries per egress port [5][19]. A single paper identified that guard bands reduce bandwidth for non-scheduled traffic and therefore proposes a heuristic to reduce guard bands [8]. Up to date, there is no scheduling algorithm that correctly handles clock synchronization errors. Moreover, processing jitter is not considered in any work about TAS scheduling and there is no algorithm in the literature that is able to maximize the bandwidth available for non-scheduled traffic. All state-of-the-art approaches providing exact results report long solving times even for medium-size problem instances.

The TAS scheduling algorithm proposed in this work is the first that copes with all mentioned challenges in a correct way using a single ILP. It is an exact method, i.e., it finds a schedule for a given problem instance if a solution exists. In the presence of multiple solutions, it proposes the one that maximizes the bandwidth left for non-scheduled traffic.

V. A Simple Fix to Handle Clock Synchronization Errors

We explain how clock synchronization errors have been handled in literature so far. We provide a small counterexample to show that the existing approach is not sufficient. Finally, we propose a simple fix against clock synchronization errors. While this fix can be easily applied in existing scheduling algorithms, we suggest a more efficient approach in Section VI-B.

A. Existing Method to Handle Clock Synchronization Errors

Some research works about scheduling for the TAS handle time synchronization errors, e.g., [6], [4], [5], and [16]. They enforce time gaps between frame transmissions for frames arriving for the same egress port. The minimum length of

these gaps is the maximum possible clock synchronization error between any two bridges or end stations. In addition, they require frame isolation (cf. Section III-A) which provides robustness against missing frames. The GCLs are computed in a postprocessing step after scheduling the transmission times of frames. The gate of a queue is opened at the time when a frame transmission is scheduled to start. The gate is closed again at the time when a frame transmission has finished[1].

B. Counterexample

The opening and closing of the gates described above is the cause why the method from the literature is not sufficiently robust against clock synchronization errors. We prove that by a small counterexample.

Assume a topology with an end station A connected via a single bridge B to an end station C. The maximum possible time synchronization error between any two devices is denoted by δ. We assume that A sends a frame f via B to C. Let A_f^B be the arrival time of frame f at bridge B according to the schedule, i.e., A_f^B is based on the reference time. We assume that the gate is open and the frame is immediately sent so that the transmission is completed at $A_f^B + d_f^{trans}$ where d_f^{trans} is the transmission duration of f. Thus, the bridge closes its gate at $A_f^B + d_f^{trans}$ according to [4]. When B's clock is before the one of A, e.g., $t_B(t) - t_A(t) = \varepsilon \leq \delta$, frame f arrives ε later at bridge B than expected as illustrated in Figure 3. Thus, it can be transmitted only by ε later than planned so that its transmission will finish only at $t_B(A_f^B + d_f^{trans}) + \varepsilon$. This is after the gate of B's egress queue has closed. As a consequence, B cannot send the frame such that its transmission is delayed until the gate opens again. This may exceed the deadline of f at its arrival at C. In a more complex setting, the delay of f causes that f will arrive late at the next bridge and possibly conflict with other frames.

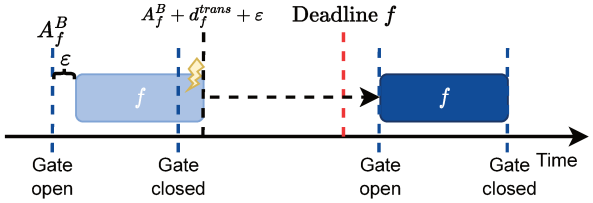


Fig. 3: Gaps between consecutive frames are not sufficient to cope with clock synchronization errors. When B's clock is ε before A's clock, frame f arrives ε later than expected at B so that f cannot be fully transmitted before the gate closes. Therefore, its transmission is delayed until the gate opens again.

C. Correction for Handling Clock Synchronization Errors

The counterexample reveals that the gate needs to be kept open until a frame's transmission has finished even if the frame arrived late. To compensate for clock synchronization error only, the gate must not be closed before $A_f^B + d_f^{trans} + \delta$. This can be added for the postprocessing that derives GCLs based on the computed schedule in [4] if the scheduler is modified such that it leaves enough space between frame transmissions.

If other sources of delay should also be respected, e.g., a frame's jitter that accumulated through processing in end stations and bridges, the safety margin between the planned transmission end and the gate closing must be larger by the maximum expected additional delay. Using a fixed upper bound for all frames is simple and may be integrated in existing scheduling algorithms. However, it may overestimate the accumulated delay of a frame, which is inefficient. Moreover, this also increases the maximum jitter of consecutively arriving frames from different ingress ports beyond δ. As a consequence, gaps between frames also need to be increased, i.e., fixing other sources of delay requires more than a safety margin before gate closings. These issues are considered in Section VI and Section VII and they are respected by the ILP in Section VII.

VI. A New Modelling Approach for TAS Scheduling

We introduce a new modelling approach for TAS scheduling from which the ILP in Section VII is derived. We formulate constraints that handle non-deterministic deviations from the schedule during execution. That means, the schedule is not corrupted if errors discussed in Section III occur as long as they remain within considered bounds.

In the following, we first explain assumptions and introduce the modelling approach. Then we propose constraints for a schedule to handle deadlines, race conditions, frame isolation, and processing jitter. In a second step, we extend the inequalities to cope with clock synchronization errors.

A. Constraints for Event Times with Uncertainty

We assume that only a single queue q per egress port is used for scheduled traffic. We will refer to it as *the queue* and to its gate as *the gate*. We model the arrival time of a frame f at node n, its variable processing time $d_{proc}^n \in [d_{proc,min}^n, d_{proc,max}^n]$ for being placed into an egress queue q, the start of its transmission on the egress port, and its transmission duration d_f^{trans}. Arrival time and transmission start are modelled as event times with uncertainty and are expressed as continuous arrival and transmission start intervals $[a_f^n, A_f^n]$ and $[t_f^q, T_f^q]$. When a schedule is executed, arrivals and transmission starts will take place within these intervals as long as perceived delays remain within assumed bounds (c.f. Figure 3). The scheduler sets earliest transmission starts t_f^q at talkers and gate closings at bridges. Gate openings are identical with earliest transmission starts at bridges so that they are also set by the scheduler. Other variables are derived from these events. We derive relevant constraints for variables in the following.

1) Latest Transmission Start: A frame arrives at a queue at the latest at $A_f^n + d_{proc,max}$. It will be sent immediately

[1] "The transformation from frame offset and queue index into gate open close events is straightforward, i.e., the offset represents the gate open event for the given queue index and the gate close event is marked by the duration of the respective frame." [4, Section 6]

when the gate is already open, or it will be sent when the gate opens at t_f^q.

$$T_f^q = \max\{A_f^n + d_{proc,max}^n, t_f^q\}. \quad (1)$$

2) Arrival Times: The transmission of a frame starts within $[t_f^q, T_f^q]$ on egress queue q attached to link l. Its duration takes d_f^{trans} time. The propagation delay is d_{prop}^l, i.e., the signal needs d_{prop}^l time to traverse l. Thus, the frame will be received at the next node n within the interval

$$[a_f^n, A_f^n] = [t_f^q + d_f^{trans} + d_{prop}^l, T_f^q + d_f^{trans} + d_{prop}^l]. \quad (2)$$

3) Race Conditions: When frames arrive at a node almost simultaneously *from different ingress ports* and have the same egress port, then their order in the egress queue may be subject to non-determinism due to hardware properties. However, for any reasonable implementation of a TSN-capable bridge, there is some minimum time difference for incoming frames on different ingress ports such that the order of frames in the egress queue is deterministic. We denote this time difference as λ.

When f_1 is scheduled to be before f_2 in the same egress queue, there must be λ time between the latest arrival plus longest processing of frame f_1 and the earliest arrival plus shortest processing of f_2:

$$A_{f_1}^n + d_{proc,max}^n + \lambda \leq a_{f_2}^n + d_{proc,min}^n \quad (3)$$

4) Frame Isolation: Frame isolation [4] denotes that any queue holds at most a single frame at a time. Let f_2 be a frame succeeding another frame f_1 in a specific queue q of node n. The frame f_1 leaves the queue at the latest at $T_{f_1}^q + d_{f_1}^{trans}$ and the frame f_2 enters it at the earliest at $a_f^n + d_{proc,min}^n$:

$$T_{f_1}^q + d_{f_1}^{trans} \leq a_{f_2}^n + d_{proc,min}^n \quad (4)$$

5) Frame Deadlines: A frame f may be multicast so that the set of its destinations is $\mathcal{M}_f = \{m_0, m_1, ...\}$. For a valid schedule, a frame f must arrive at all its destinations before its deadline D_f:

$$\forall m \in \mathcal{M}_f : \; A_f^m \leq D_f. \quad (5)$$

6) Gate Closings: We consider constraints for a gate closing event at the time t_{close}^q. Let f_{prev} be the last frame sent before the gate closed, and f_{next} the first frame sent after the gate closed and opened again.

The gate must close late enough so that the previous frame has been fully sent.

$$T_{f_{prev}}^q + d_{f_{prev}}^{trans} \leq t_{close}^q \quad (6)$$

The gate must close early enough so that the next frame cannot be sent when it arrives at the earliest possible time at the egress queue.

$$t_{close}^q \leq a_{f_{next}}^n + d_{proc,min}^n + d_{f_{next}}^{trans} \quad (7)$$

Figure 4 illustrates the connection between time variables and parameters. In this specific example, the earliest transmission start is the time after the frame arrives at egress queue q, so it is queued for some time. The latest arrival time at the queue is at $A_f^n + d_{proc,max}^n$ when the gate is already open for some time.

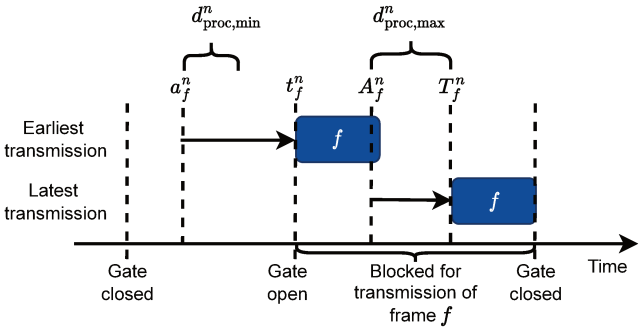


Fig. 4: A frame f arrives within $[a_f^n, A_f^n]$ and its transmission starts within $[t_f^n, T_f^n]$. Clock synchronization errors are not considered. The gate must be open long enough so that the frame can be sent under all conditions.

To reserve the egress port for the transmission of frame f, the scheduler blocks the interval $[t_f^n, T_f^n + d_f^{trans}]$ between the frame's earliest transmission start t_f^n and the end of its latest transmission for the transmission of other frames.

The difference between earliest and latest transmission starts $T_f^n - t_f^n$ may increase with every hop. However, the difference may also be reduced if a frame is scheduled to wait at a closed gate.

B. Dealing with Synchronization Errors

We first introduce the concept of reference time and local time. Then, we augment the previous constraints by considerations for clock synchronization errors.

1) Reference Time and Local Time: Schedules are expressed relative to a reference time t. This includes intervals for frame arrivals and transmission starts, gate openings and closings.

However, devices have local clocks that may deviate from the reference time. Let δ be the maximum clock synchronization error, i.e., the maximum difference between any pair of clocks. End stations and bridges execute the schedule according to their local clock, i.e., transmission starts at end stations and GCL entries at bridges. Moreover, deadlines are interpreted from the perspective of a receiving end station.

2) Race Conditions: Equation (3) is modified assuming that frame f_2 is δ time earlier than expected:

$$A_{f_1}^n + d_{proc,max}^n + \lambda \leq a_{f_2}^n + d_{proc,min}^n - \delta \quad (8)$$

3) Frame Isolation: Equation (4) is modified assuming that frame f_2 was sent δ time earlier than expected:

$$T_{f_1}^q + d_{f_1}^{trans} \leq a_{f_2}^n + d_{proc,min}^n - \delta \quad (9)$$

4) Frame Deadlines: Equation (5) is modified assuming that frame f was sent by the penultimate hop δ time later than expected:

$$\forall m \in \mathcal{M}_f : \; A_f^m + \delta \leq D_f. \tag{10}$$

5) Gate Closings: We modify Equation (6) by considering that the previous frame is δ time later than expected:

$$T_{f_{prev}}^q + d_{f_{prev}}^{trans} + \delta \leq t_{close}^q. \tag{11}$$

Likewise, we modify Equation (7) by considering that the next frame is δ time earlier than expected:

$$t_{close}^q \leq a_{f_{next}}^n + d_{proc,min}^n + d_{f_{next}}^{trans} - \delta. \tag{12}$$

VII. Scheduling Algorithm

We describe an ILP model used for the computation of robust schedules. First, we give a brief introduction to ILPs. Then, we summarize the modelling approach of the ILP. Afterwards, we introduce three scheduling algorithms based on the ILP model which correspond to different assumptions about schedule robustness. Finally, we formally state the ILP model.

A. Fundamentals of ILPs

An ILP describes the solution space to a problem with linear inequalities. Every assignment of variables which fulfills all inequalities corresponds to a solution of the problem and vice versa. Some variables may be restricted to take only integer values. A linear objective function may be used to choose the best solution. An ILP solver implements algorithms to find a feasible variable assignment which minimizes or maximizes the objective function. ILP solving is NP-complete, i.e., finding a feasible solution to an ILP requires exponential work as a function of the input size in the worst case. However, state-of-the-art solvers are able to find and optimize solutions for reasonably sized problem instances in many cases. We use the solver Gurobi [20] in the remainder of this paper.

B. Modelling Approach

We summarize the ideas needed to assess the evaluation results.

1) Handling Streams with Different Periods: The period of a stream s is denoted as p_s. Every stream s has an earliest and latest time E_s and L_s when the transmission of s must start at the talker of s. Likewise, every stream s has a deadline D_s until its payload must have arrived at all its destinations.

Streams may have different periods. The hyperperiod H is the least common multiple of all stream periods. The hyperperiod is the duration of the periodic schedule. The ILP requires that all streams have period H. When streams have different period, a stream with a period $p_s < H$ is modelled as a composite of $\frac{H}{p_s}$ stream instances s_i, $0 \leq i < \frac{H}{p_s}$. Each stream instance s_i has period H. Its earliest and latest transmission starts and its deadline are shifted copies of the original stream: $D_{s_i} = D_s + i \cdot p_s$, $E_{s_i} = e_s + i \cdot p_s$, and $L_{s_i} = L_s + i \cdot p_s$. Thus, the deadline of a frame f belonging to stream instance s_i is $D_f = D_{s_i}$. Similarly, earliest and latest transmission times of a frame from its talker are $E_f := E_{s_i}$ and $L_f := L_{s_i}$.

2) Gate Closings: A central innovation of the ILP is the modelling of gate closings. Let f_{prev} and f be two frames transmitted successively via egress queue q. We use a single binary variable z_f^q indicating whether some interval in $[T_{f_{prev}}^q + d_{f_{prev}}^{trans}, t_f^q]$ is released for other traffic. If and only if the variable is set to 1, the gate for scheduled traffic is closed in this interval and opened again at t_f^q. With the help of these variables the number of gate closing events and thereby GCL entries within a hyperperiod can be limited. Furthermore, this indication is used to compute the bandwidth released for non-scheduled traffic.

Most works from literature do not consider GCL synthesis at all. The few models considering GCL synthesis take a different approach. The number of time interval between gate openings and closings are fixed in advance in [5] and [19]. A binary variable is used for every frame at every hop for every time interval. Therefore, the model size increases not only with the number of frames, but also with the number of available GCL entries. This results in an exponential growth of the solution space in the number of GCL entries. In contrast, the model size of presented approach is independent of the number of GCL entries.

3) Objective Function: The TAS is designed such that bandwidth can exclusively be reserved for scheduled and non-scheduled traffic, respectively. Maximizing bandwidth remaining for non-scheduled traffic while accommodating all scheduled streams seems a reasonable objective function. GCL entries can be used to release more time intervals for non-scheduled traffic, but result in guard bands which occupy bandwidth. Thus, there is a non-trivial trade-off when this objective is maximized. The time remaining exclusively for non-scheduled traffic is the time with a closed gate for scheduled traffic minus the time intervals for guard bands. This time is summed up in the ILP for all egress ports with the help of the above mentioned variables z_f^q and used as objective function to maximize. We remark that this is the first work about scheduling in TSN which applies this objective function and which handles guard bands during scheduling.

C. Scheduling Algorithms

With subsets of constraints given for the ILP in Section VII-D, three different kinds of schedules can be computed.

1) Tight Schedules (TS): TS implement frame isolation to cope with missing frames, but they do not consider race conditions, processing jitter, and clock synchronization errors. That means, Equations 16 – 17, 19, 32, and 34 – 36 are not contained in the ILP. All occurrences of variables for latest transmission starts T_f^q are replaced by the corresponding earliest transmission start t_f^q. Time synchronization error δ and race condition gaps λ are removed from Equations 22, 25, and 38.

2) Naïve Robust Schedules (NRS): NRS are robust with respect to all considered sources of non-determinism mentioned in Section III. They are constructed by the ILP restricted to

the Equations 14 – 15, 18 – 31, 33, 38. All occurrences of T_f^q are replaced by t_f^q. To achieve robustness, a minimum gap between consecutive frame transmissions and between frame transmissions and gate closings is ensured (Equations 39 – 44). This minimum gap is chosen large enough that frame transmissions cannot interfere which each other or gate closings regardless of processing jitter or time synchronization errors, which mostly overestimates the necessary gap. Let $\mathcal{N}_f^{max}$ be the set of nodes along the longest path of a stream. The minimum length of a gap must at least be

$$\lambda + \sum_{n \in \mathcal{N}_f^{max}} d_{proc,max}^n - d_{proc,min}^n. \tag{13}$$

NRS are similar to schedules from state-of-the-art scheduling algorithms if they are fixed for robustness by our recommendations in Section V-C where the gaps are enlarged to cover processing jitter.

3) Efficient Robust Schedules (ERS): ERS are also robust against all sources of non-determinism mentioned in Section III. They are computed using all constraints for the ILP in Section VII-D using the event times with uncertainties derived in Section VI. In contrast to NRS, the gaps between consecutive frame transmissions and between frame transmissions and gate closings have variable size to be just large enough.

4) Comparison: TS, NRS, and ERS utilize the same objective function, i.e., they maximize the bandwidth usable for non-scheduled traffic. If a problem instance can be scheduled with NRS, it can also be scheduled with ERS and TS, and if a problem instance can be scheduled with ERS, it can also be scheduled with TS. However, the resulting schedules are substantially different as TS have least space between frame transmissions, followed by ERS, and NRS providing most space between frame transmissions.

D. ILP Model

We elaborate on the details of the proposed ILP model as a reference for implementation. However, we remark that the highlighted modelling approaches in Section VII-B are sufficient to asses the evaluation results. First, we introduce the parameters and variables of the model. Then, we formally state the linear constraints and the objective. Afterwards, we highlight the slight differences of the models for NRS, ERS, and TS. Finally, we discuss performance considerations with respect to solving the model.

1) Nomenclature: We remark that we reuse the variables and parameter symbols from Section VI with the same semantics.

a) Parameters: Let $\mathcal{N}$ and $\mathcal{L}$ denote the set of nodes and links, respectively. A node is either an end station or a bridge. A bridge has minimum and maximum processing delays. For every node n, we denote these delays as $d_{proc,min}^n$ and $d_{proc,max}^n$. The largest possible gap between the clocks of any pair of nodes is denoted by δ. The time gap between frame arrivals such that no race conditions for the same egress queue can occur it denoted by λ. Links are modeled as unidirectional connections from an egress port of one device to an ingress port of another device. Every link l has a specific propagation delay d_{prop}^l. The egress port of an egress queue q has a specific transmission rate r_{trans}^q. Thus, the duration of a guard band for the respective egress port is $t_{GB}^q := \frac{1542\,B}{r_{trans}^q}$. The number of available GCL entries in this egress port is n_{GCL}^q. Every frame f has a deadline D_f, an earliest transmission offset E_f, and a latest transmission offset L_f, as defined in Section VII-B1. We consider every stream instance as a separate stream in the ILP model for the sake of readability, i.e., we omit stream instance indices. The size of a frame f in byte is b_f.

b) Variables: Variables are used to represent a schedule. Transmission offsets are times at which frames are sent from an end station or bridge. Processing jitter leads to jitter of transmission offsets of frames. Therefore, exact transmission times when a schedule is executed are uncertain during scheduling. We use variables to capture the earliest and the latest possible transmission starts of frames. The earliest possible transmission offset of frame f at the egress queue q is denoted by t_f^q. The respective latest possible transmission offset of the same frame is captured by the variable T_f^q. All transmission offsets are meant to be relative to the start of a hyperperiod, i.e., the start of the schedule. To ensure that transmissions over the same egress queue q cannot overlap, we introduce binary variables $o_{f,f'}^q$ for every two frames $f \neq f'$. This variable is set to 0 if f is sent before f' at q, and 1 otherwise. Similarly, the arrival order of frames arriving from different ingress links l and l' attached to the egress queues q, q' and forwarded from the same next egress port is captured by $o_{f,f'}^{q,q'}$.

When a frame is dispatched and sent from an egress queue, at least one of the following conditions must hold:

1) The frame is sent immediately after it was processed as the gate is open.
2) The frame waits until the gate is opened and transmission starts immediately after opening.

In case (2), the gate must have been closed after the transmission of the previous frame and is opened again before the frame is dispatched. This implies that 2 GCL entries are needed every time case (2) is used in a schedule. The binary variable z_f^p is set to 1 when case (2) is used prior to the transmission of a frame f at egress queue q. We use the variable c_f^q for the closing time of the gate between the transmission of f and the transmission of the previous frame. The binary variable z_{last}^q captures whether the gate is closed in the last time interval after all frame transmissions of q. We use the continuous variable c_{last}^q for the closing time of the gate after all frame transmissions at q. By subtracting time intervals with an open gate and guard bands from the hyperperiod, the time available for other traffic classes can be computed from these variables (cf. Equation 37).

2) Constraints: This section describes the constraints that must hold for every valid schedule. We present their intuitive meaning and how they are expressed by the ILP.

a) Transmission Offset: Let f be a frame and q be the first egress queue on the path of f. The transmission from the

source of the stream must be in the range between earliest and latest transmission offset:

$$E_f \leq t_f^q \leq L_f. \tag{14}$$

Furthermore, if f and f' are the same frame of the i-th and i+1-th stream instances of the same stream s, and q is the egress queue of the source node of s, then their transmission offsets must be equal relative to their respective periods. Thus, it must hold:

$$t_f^q + p_s = t_{f'}^q \tag{15}$$

$$T_f^q + p_s = T_{f'}^q. \tag{16}$$

b) Per-Frame Jitter: The latest possible transmission offset of a frame at egress queue q is always later than or at the same time as the respective earliest possible transmission offset (Equation 17).

$$t_f^q \leq T_f^q \tag{17}$$

c) Forwarding After Arrival: Before a bridge can forward a frame, the frame must have arrived at the bridge. Transmission, propagation, and processing delay must also be considered. This must hold for the minimum and maximum processing delay (Equations 18 and 19). Let f be a frame and q_1, q_2 be two consecutive egress queues on the path of f. Furthermore, let l_1 be the links attached to q_1 and n_2 be the device of q_2. The following inequalities enforce that frames are forwarded after arrival:

$$t_f^{q_1} + d_{prop}^{l_1} + d_{proc,min}^{n_2} + \frac{b_f}{r_{trans}^{q_1}} \leq t_f^{q_2} \tag{18}$$

$$T_f^{q_1} + d_{prop}^{l_1} + d_{proc,max}^{n_2} + \frac{b_f}{r_{trans}^{q_1}} \leq T_f^{q_2}. \tag{19}$$

.

d) Deadline: All frames of a stream must arrive at their destinations before their deadlines. Let q be the egress queue of a last hop of a frame f. Let l be the link attached to q. As only transmission offsets are used as variables, the respective delays caused by the last hop must be considered. The following inequality ensures that the deadline is met even for the latest possible transmission offset:

$$T_f^q + d_{prop}^l + \frac{b_f}{r_{trans}^q} \leq D_f - \delta. \tag{20}$$

As streams may be multicast, this constraint must hold for all destinations of a stream.

e) Non-Overlapping Transmission: Two frames f_1, f_2 cannot be in transmission over the same egress queue q at the same time. This must hold for the earliest and latest possible transmissions and for time synchronization errors. The following constraints enforces this with the binary order variables o_{f_1,f_2}^q:

$$T_{f_1}^q + \frac{b_{f_1}}{r_{trans}^q} - M \cdot o_{f_1,f_2}^q + \delta \leq t_{f_2}^q + \frac{b_{f_2}}{r_{trans}^q}. \tag{21}$$

M is some arbitrary constant which is greater than the largest possible gap between both transmission offsets. The hyperperiod places an upper bound on M, but smaller values should be used when available. If o_{f_1,f_2}^q is set to 0, the term $-M \cdot o_{f_1,f_2}^q$ is 0 and the transmission of f_1 must be finished before the transmission of f_2 starts. Otherwise, this term effectively turns off the constraint as the left-hand side takes a negative value. This approach is denoted as Big M method in the optimization literature and is used in various other constraints of the presented ILP.

f) Isolation: Two frames f_1, f_2 cannot be in the same egress queue at the same time (Equation 22). However, both frames must be isolated even longer in time to mitigate unsynchronized clocks. The earliest possible arrival of the arriving frame must be after the latest possible transmission of the previous frame. Let q_1 be the egress queue attached to the ingress link l_1 on the path of f_1 to some bridge n_2, q_2 be the next egress queue of f_1, and f_2 be another frame forwarded over q_2. Isolation is ensured by the following constraint:

$$\begin{aligned} &T_{f_1}^{q_2} + \frac{b_{f_2}}{r_{trans}^{q_2}} - M \cdot o_{f_1,f_2}^{q_2} + \delta \\ &\leq t_{f_2}^{q_1} + d_{prop}^{l_1} + d_{proc,min}^{n_2} + \frac{b_{f_2}}{r_{trans}^{q_1}}. \end{aligned} \tag{22}$$

g) Frame Order: When the message of a stream is too large to be sent in one frame, multiple frames are used. These frames must arrive in-order, so they must be sent in-order (Equation 23). Let f_1, f_2 be two frames of the same stream s such that f_1 and f_2 belong to the same stream and are consecutive frames of the same message, and q be an egress queue on the path of s. The following equation enforces frame order indirectly through Equation 21:

$$o_{f_1,f_2}^q = 0. \tag{23}$$

h) FIFO: Frames arriving at an egress port must be forwarded in the same order. If two frames f_1, f_2 share two consecutive egress ports on their respective paths, this can be enforced by requiring the same transmission order for both egress ports (Equation 24). Let q_1, and q_2 be consecutive egress queues on the paths of the frames f_1, f_2. The following equation enforces this in combination with Equation 21:

$$o_{f_1,f_2}^{q_1} = o_{f_1,f_2}^{q_2}. \tag{24}$$

When two frames arrive at different ingress ports and share the same egress port for their next hop, the ordering of arrivals must also be considered. These arrivals are isolated by $\lambda + \delta$ to prevent race conditions and time synchronization errors (Equation 25). Let q_1 and q_2 be the egress queues attached to two ingress links l_1, l_2 to some bridge and q_3 an egress queue of this bridge. Furthermore, let f_1 be a frame which is transmitted consecutively over q_1 and q_3, and q_2 be a frame which is transmitted consecutively over q_2 and q_3. Transmission order of f_1 and f_2 is preserved by the following constraint:

$$\begin{aligned} &T_{f_1}^{q_1} + \frac{b_{f_1}}{r_{trans}^{q_1}} + d_{prop}^{l_1} - M \cdot o_{f_1,f_2}^{q_3} + \lambda + \delta \\ &\leq t_{f_2}^{q_2} + \frac{b_{f_2}}{r_{trans}^{q_2}} + d_{prop}^{l_2}. \end{aligned} \tag{25}$$

i) Gates and GCLs: Let q be an egress queue and f be a frame forwarded over q. The gate for scheduled traffic can only be closed between the transmissions of two consecutive frames of scheduled traffic. Therefore, we model GCL entries by variables z_f^q indicating whether the gate is closed before the transmission of f an when it was closed c_f^q. If the gate was closed, it is opened at the earliest possible transmission offset of f, t_f^q. The minimum duration of a time interval is the duration of a guard band due to technical requirements (Equation 26). Vice versa, we set the duration of the corresponding not used time interval to 0 when the gate was not closed (Equation 27). The closing of a gate must be before its matching opening (Equation 28). Additionally, the closing must be after the latest possible transmission ends for every frame f_0 sent before the transmission of frame f (Equation 29). Otherwise, an earlier frame may move to another time interval when it is delayed. The number of GCL entries of the egress port of q is limited by n_{GCL}^q (Equation 30). Two entries are consumed for every closing and opening of the gate.

$$t_{GB}^q \cdot z_f^q \leq t_f^q - c_f^q \tag{26}$$

$$t_f^q - c_f^q \leq M \cdot z_f^q \tag{27}$$

$$c_f^q \leq t_f^q \tag{28}$$

$$T_{f_0}^q + \frac{b_{f_0}}{r_{trans}^q} - M \cdot o_{f_0,f}^q + \delta \leq c_f^q \tag{29}$$

$$\sum_{f \text{ forwarded over } q} z_f^q \leq \frac{n_{GCL}^q}{2} \tag{30}$$

Constraints similar to Equations 26–30 must hold for z_{last}^q and c_{last}^q. Finally, the forwarding of a frame depends on the state of the gate when the frame arrives at an egress queue. If the gate is open, the frame is transmitted without queuing delay. Let q_0 be the egress queue attached to the ingress link l_0 before q on the path of f. Furthermore, let n be the node of q. Equations 31 and 32 enforce immediate forwarding when the gate is open:

$$t_f^q \leq t_f^{q_0} + d_{prop}^{l_0} + d_{proc,min}^n + \frac{b_f}{r_{trans}^{q_0}} + M \cdot z_f^q \tag{31}$$

$$T_f^q \leq T_f^{q_0} + d_{prop}^{l_0} + d_{proc,max}^n + \frac{b_f}{r_{trans}^{q_0}} + M \cdot z_f^q. \tag{32}$$

If GCL entries are used to close and open the gate before the transmission of f over q, the gate must be closed when f arrives over l_0 (Equation 33). Otherwise, f would be forwarded without queuing delay which contradicts the minimum size of a time interval (Equation 26).

$$\begin{aligned} &c_f^q - M \cdot \left(1 - z_f^q\right) + \delta \\ \leq &t_f^{q_0} + d_{prop}^{l_0} + d_{proc,min}^n + \frac{b_f}{r_{trans}^{q_0}} \end{aligned} \tag{33}$$

j) Latest Possible Transmission: There are two cases how a frame is forwarded when a frame arrives at the latest possible time. If the gate is closed at the latest possible arrival time of a frame, the frame is sent when the gate is opened again. The latest possible transmission offset equals the gate open time as well as the earliest possible transmission offset in this case. Otherwise, the frame is dispatched immediately without queuing delay. We use a binary variable y_f^q to enforce one of these cases for frame f sent from egress queue q of node n. The variable is set to 1 if and only if the latest transmission starts immediately after processing. Let q_0 be the egress queue attached to the ingress link l_0 on the path of f before q. Exactly one of the cases is enforced by Equations 34 – 36:

$$T_f^{q_0} + d_{prop}^{l_0} + d_{proc,max}^n + \frac{b_f}{r_{trans}^{q_0}} \leq t_f^q + M \cdot y_f^q \tag{34}$$

$$T_f^q - M \cdot y_f^q \leq t_f^q \tag{35}$$

$$T_f^q \leq T_f^{q_0} + d_{prop}^{l_0} + d_{proc,max}^n + \frac{b_f}{r_{trans}^{q_0}} + M \cdot \left(1 - y_f^q\right). \tag{36}$$

3) Objective: We maximize the time available for other traffic classes per egress port (Equation 37) as we want to analyze the usage of resources in TSN. This time is identical to the times when the gates of queues dedicated for scheduled traffic are closed, minus the corresponding guard bands. Therefore, we add up these times for all egress queues. Let $\mathcal{Q}$ denote the set of all egress queues. The following term formally states the objective:

$$\begin{aligned} \max \sum_{q \in \mathcal{Q}} &H - c_{last}^q - t_{GB}^q \cdot z_f^q \\ &+ \sum_{f \text{ forwarded over } q} t_f^q - c_f^q - t_{GB}^q \cdot z_f^q. \end{aligned} \tag{37}$$

a) Redundant Constraints: The following constraints are not necessary for the correctness of the model. However, they enable the ILP solver to infer tighter bounds for the objective function which speeds up solving. The time available to other traffic classes is bounded by the remaining time after subtracting transmission durations of scheduled traffic frames, and times which are wasted to cope with time synchronization errors and processing delay jitter (Equation 38). Let q be an egress queue. The additional constraint is given in the following:

$$\begin{aligned} &H - c_{last}^q + \sum_{f \text{ forwarded over } q} t_f^q - c_f^q \\ \leq &H - \left(\sum_{f \text{ forwarded over } q} \frac{b_f}{r_{trans}^q} + \delta + \left(T_f^q - t_f^q\right) \right). \end{aligned} \tag{38}$$

4) Modifications for NRS: NRS enforces gaps between frame transmissions and gate closings to ensure robustness. These gaps are long enough to compensate for processing jitter, time synchronization errors, and race conditions. Let G be the duration of the minimum gap to ensure deterministic behavior. ILPs for NRS are constructed with Equations 14 – 15, 18 – 31, 33, 38. All occurrences of T_f^q are replaced by t_f^q for all frames f and egress queues q. All occurrences of δ

and/or λ are replaced by G. For instance, Equation 21 ensuring exclusive link usage is replaced by

$$t_{f_1}^q + \frac{b_{f_1}}{r_{trans}^q} - M \cdot o_{f_1,f_2}^{q_1} + G \leq t_{f_2}^q + \frac{b_{f_2}}{r_{trans}^q}. \tag{39}$$

Similarly, frames must arrive G time before their deadline and Equation 20 is modified as follows:

$$t_f^q + d_{prop}^l + \frac{b_f}{r_{trans}^q} \leq D_f - G. \tag{40}$$

The constraint which enforces isolation (Equation 22) enforces the gap G between arrivals and transmissions:

$$\begin{aligned} & t_{f_1}^{q_1} + \frac{b_{f_1}}{r_{trans}^{q_1}} - M \cdot o_{f_1,f_2}^{q_1} + G \\ & \leq t_{f_2}^{q_2} + d_{prop}^{q_2} + d_{proc,min}^{n_1} + \frac{b_{f_2}}{r_{trans}^{q_2}}. \end{aligned} \tag{41}$$

Equation 33 for gaps between frame transmissions and gate closings is modified in the following way:

$$\begin{aligned} & c_f^q - M \cdot \left(1 - z_f^q\right) + G \\ & \leq t_f^{q_0} + d_{prop}^{l_0} + d_{proc,min}^{n} + \frac{b_f}{r_{trans}^{q_0}}. \end{aligned} \tag{42}$$

Equation 25 preventing race conditions is replaced by

$$\begin{aligned} & t_{f_1}^{q_1} + \frac{b_{f_1}}{r_{trans}^{q_1}} + d_{prop}^{l_1} - M \cdot o_{f_1,f_2}^{q_3} + G \\ & \leq t_{f_2}^{q_2} + \frac{b_{f_2}}{r_{trans}^{q_2}} + d_{prop}^{l_2}. \end{aligned} \tag{43}$$

Finally, the redundant constraint in Equation 38 is modified for NRS:

$$\begin{aligned} & H - c_{last}^q + \sum_{f \text{ forwarded over } q} t_f^q - c_f^q \\ & \leq H - \left(\sum_{f \text{ forwarded over } q} \frac{b_f}{r_{trans}^q} + G \right). \end{aligned} \tag{44}$$

5) Performance Considerations: The number of constraints grows linearly with the length of paths of the streams, and thus indirectly with the number of bridges. Moreover, the ILP grows quadratically with the number of streams and frames in the worst case. Current state-of-the-art ILP solvers offer interfaces to integrate custom heuristics for solution construction and refinement. We implemented a custom heuristic to quickly find initial feasible solutions. The heuristic sets all gates to be never closed, i.e, fixing all variables z_f^q to 0, and solves the restricted simpler ILP.

VIII. Evaluation

We argued that TS are most efficient, followed by ERS and NRS. However, TS may fail on real hardware bridges due to processing jitter, race conditions, and clock synchronization errors while NRS and ERS are designed to be robust against these issues.

Thus, we pursue three key questions in this evaluation. What is the price of robustness, i.e., what is the performance gap between TS and NRS or ERS? What is the benefit of ERS vs. NRS? And is the computation of ERS scalable compared to other TSN scheduling algorithms?

We first introduce the methodology. Then, we analyze how bandwidth is utilized by the three schedule types depending on many factors. We study the impact of the schedule type on the number of admissible streams which can be considered as a key performance indicator. Finally, we compare the runtime of the algorithm to compute ERS from this paper with four well-known approaches from the literature.

A. Methodology

A problem instance consists of a network, a set of TT streams to be scheduled, and a bridge model including non-determinism. For every problem instance TS, NRS, and ERS are computed and analyzed for a direct and fair comparison. In the following, we present the network model, the bridge model including non-determinism, and the traffic model, and we explain how schedules are computed.

1) Network Model: The networks in our study are rings, where each node is connected to four end stations. Unless otherwise stated, the rings have 10 bridges. This is a realistic model as rings are common in factory automation use cases of TSN [9][21][22][23]. Although rings are simple topologies, they constitute worst cases for TAS scheduling as many streams compete for the bandwidth of a few links. Traffic forwarding follows shortest paths that are computed by Dijskstra's algorithm. The transmission capacity of all links l is 1 Gb/s and their propagation delay is $d_{\text{prop}}^l = 0.1\,\mu s$; the latter corresponds to a cable length of 20 m.

2) Bridge Model Including Non-Determinism: If not stated otherwise, the number of available GCL entries per egress port is $n_{\text{GCL}} = 512$. We use this large number as default to avoid that the impact of short GCLs dominates the impact of the schedule type.

The mean processing delay of the bridges is $1.55\,\mu s$ with a maximum symmetric jitter of $0.3\,\mu s$ by default. That means that the minimum and maximum processing delays are $d_{\text{proc,min}} = 1.4\,\mu s$ and $d_{\text{proc,max}} = 1.7\,\mu s$. We obtained these values for current TSN bridges from expert knowledge.

To prevent race conditions on an egress queue, a minimum time gap of $\lambda = 0.4\,\mu s$ is needed for the arrival of frames from different ingress ports.

The maximum clock synchronization error between any two devices is $\delta = 1\,\mu s$.

3) Traffic Model: We generate random streams by selecting their source and destination from the set of all end stations where source and destination are different. The payload of a stream is delivered by a single data frame with random size in the range $[84\,B, 1542\,B]$. This range corresponds to transmission durations in the range $[0.672\,\mu s, 12.336\,\mu s]$. We randomly select the period p_s of a stream s from the set $\{500\,\mu s, 750\,\mu s, 1500\,\mu s\}$. These values are in the range of isochronous traffic for industrial automation streams [24].

	NRS					ERS					TS				
GCL entries	Used entries	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)	Used entries	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)	Used Entries	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)
1	1	0.411	0	88.49	11.07	1	0.411	0	88.49	11.07	1	0.411	0	88.49	11.07
2	2	0.822	27.6	60.5	15.47	2	0.822	28.5	59.6	15.66	2	0.822	29.6	58.5	15.92
4	4	1.64	51	36.3	23.4	4	1.64	53.9	33.4	24.91	4	1.64	56.9	30.4	26.7
8	7.989	3.29	66.5	19.1	36.69	7.994	3.29	70.8	14.8	42.81	7.989	3.29	74.6	11.1	50.01
16	15.26	6.28	73	9.63	53.47	15.34	6.31	76.3	6.31	63.71	15.08	6.2	79.2	3.52	75.85
32	20.35	8.37	73.7	6.86	61.73	20.46	8.41	76.8	3.69	75	19.19	7.89	79.4	1.6	87.35
64	20.44	8.41	73.7	6.78	62.03	20.59	8.47	76.8	3.64	75.28	19.53	8.03	79.4	1.46	88.35

Tab. 1: Use of GCL entries and bandwidth depending on the number of available GCL entries per egress port. 100 streams are scheduled with NRS, ERS, and TS on the same problem instances. These 100 streams occupy 11.1% of the bandwidth. Legend: GB = guard band, ρ = utilization of bandwidth reserved for scheduled traffic.

This model implies that the hyperperiod of all streams is at most $H = 1500\,\mu s$, and a problem instance with k streams yields on average $2 \cdot k$ stream instances in a schedule.

The start of the periods of all streams is synchronized, and the earliest transmission start is the beginning of the period. The latest transmission start and the deadline are the end of the period. This assumption is common in literature [3] and constitutes a worst case for scheduling.

4) Schedule Synthesis: For every problem instance, we construct ILPs to compute the different schedule types NRS, ERS, and TS as outlined in Section VII-C. We use Gurobi 10 [20] with a plugin for the custom heuristic from Section VII-D5 to solve the constructed ILPs. All evaluations were performed on a system equipped with a Ryzen 3900X with $12 \times 3.80\,\text{GHz}$ cores and $64\,\text{GB}$ RAM.

The computation of NRS requires a minimum gap between frames according to Equation 13. In a ring with 10 bridges the longest path contains 6 bridges. Thus, the minimum gap is

$$\begin{aligned}&\delta + \sum_{n \text{ on longest stream path}} d^n_{\text{proc, max}} - d^n_{\text{proc, min}}\\ =&1\,\mu s + 6 \cdot 0.3 \mu s = 2.8\,\mu s\end{aligned}$$

This compares to an average frame transmission duration of $6.5\,\mu s$.

B. Bandwidth Usage

We analyze how bandwidth is used by different schedule types. We argue that the utilization of bandwidth reserved for scheduled traffic is useful for comparisons. We investigate how this metric is influenced by network and traffic parameters, as well as by different causes of non-determinism at scheduling time.

1) Classification of Bandwidth Usage: A periodic schedule divides the time into periodic intervals with open gates for queues with scheduled and non-scheduled traffic. We denote the percentage of bandwidth used for purpose X by b_X. The times with open gates for non-scheduled traffic subdivides into times for guard bands (b_{GB}) and times when non-scheduled traffic can be sent (b_{NST}). The times with open gates for scheduled traffic can be subdivided into times when scheduled traffic is sent (b_{ST}) and into slack time (b_{slack}). While the exact intervals for transmissions and slack cannot be known a priori, the bandwidth on the physical layer of the scheduled traffic is known, which allows computation of b_{ST}. The utilization of bandwidth reserved for scheduled traffic is:

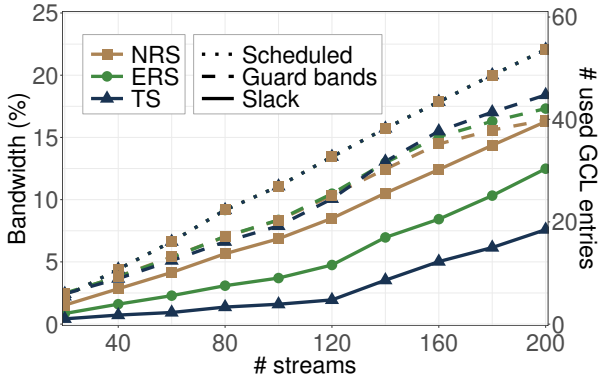
$$\rho = \frac{b_{ST}}{b_{ST} + b_{slack}}. \tag{45}$$

Slack time consists of gaps between frames or between frames and gate closings in NRS and ERS. However, there may also be unused time between transmissions of scheduled frames that is just too short, i.e., shorter than a guard band, to open the queue for non-scheduled traffic. This type of slack can also be found in TS.

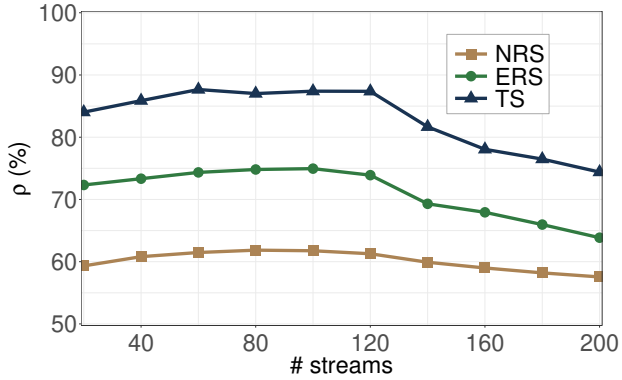
2) Experiment Setup: In the following, we perform experiments that differentiate in some parameters from the presented problem instances. Per experiment, we sample 20 problem instances as outlined in Section VIII-A, each with 100 random streams. We compute NRS, ERS, and TS for these instances. We set a time limit of $1\,h$ per computation and report the best solution found by then. We analyze how different schedules utilize bandwidth according to Section VIII-B1. To that end, we average the results of all links on the ring and of all instances, and report these average values per schedule type.

3) Impact of GCL Entries: In the first experiment, we study the impact of available GCL entries per egress port on bandwidth usage. Table 1 compiles the results depending on the number of available GCL entries. Performance metrics are shown for all three schedule types.

The maximization objective is the remaining bandwidth for non-scheduled traffic. It increases with the number of available GCL entries. When more available GCL entries enable more alternating intervals for scheduled and non-scheduled traffic, schedules provide more bandwidth for non-scheduled traffic as this allows usage of sufficiently large gaps between scheduled frames for non-scheduled traffic. This also decreases slack. Therefore, the utilization of bandwidth reserved for scheduled traffic also increases with more available GCL entries. Thus, sufficient GCL entries are crucial that bandwidth can be efficiently utilized for both scheduled and non-scheduled traffic. To avoid that the number of available GCL entries limits the performance of the scheduling algorithms, we allow 512 GCL entries per egress port in all subsequent evaluations. However, from 32 GCL entries on, the bandwidth for non-scheduled traffic and the number of used GCL entries hardly increase. In fact, the largest number of used entries we have seen in all problem instances was 38.



(a) Bandwidth for scheduled traffic, guard bands and slack per egress port. Bandwidth for guard bands and number of used GCL entries are show by identical lines.



(b) Utilization of bandwidth reserved for scheduled traffic ρ.

Fig. 5: Use of GCL entries and bandwidth depending on the number of scheduled streams for NRS, ERS, and TS.

Scheduled traffic occupies on average 11.1% of the bandwidth when 100 random streams are scheduled, which is the same in all experiments of Table 1. Therefore, this percentage is not indicated in the table itself. The number of guard bands per hyperperiod is half the number of utilized GCL entries. Therefore, the bandwidth used for guard bands scales linearly with the number of used GCL entries. When 32 or more GCL entries are available, the bandwidth for guard bands is almost as large as the bandwidth for scheduled traffic.

With 20.44 guard bands per link on average and 20 links in the bidirectional ring, 408.8 GCL entries are utilized in the entire network, which results into 204.4 intervals reserved for scheduled traffic. 100 streams with 2.5 hops on the link and 2 frames per hyperperiod on average amount for 500 frames on the links. Thus, $\frac{500}{204.4} = 2.45$ frames are sent on average within a reserved interval. Only the first frame may be queued by a closed gate, the other frames are immediately forwarded. Queuing a frame for very short time has the potential to reduce its jitter.

While these findings qualitatively hold for all three schedule types, their results slightly differ in absolute numbers. NRS reveal most slack and have room for least non-scheduled traffic while TS show least slack and most non-scheduled traffic. However, these numbers do not deviate a lot because they are limited by the moderate number of scheduled streams. In contrast, the utilization of bandwidth reserved for scheduled traffic significantly differs for NRS, ERS, and TS. This underlines that ERS accommodates scheduled streams with less slack than NRS, which is more efficient with regard to reserved bandwidth. TS are even more efficient than ERS, but they are not recommendable in practice as they are not robust.

4) Impact of Scheduled Streams: We vary the number of scheduled streams and investigate their influence on GCLs and bandwidth usage. Figure 5(a) shows that the average number of guard bands scales about linearly with the number of streams. The number of used GCL entries is a linear function of the bandwidth needed for guard bands and is shown by the lines for guard bands when the second y-axis is applied. That means, intervals reserved for scheduled traffic hold about 2.5 frames regardless of the number of scheduled streams.

The amount of bandwidth needed for guard bands is about the same for all considered schedule types. 200 streams require between 15% and 18% of the bandwidth for guard bands.

Figure 5(a) further illustrates that the bandwidth occupied by scheduled traffic linearly increases with the number of scheduled streams. It is 22.06% for 200 streams. The reservations for scheduled traffic also contain slack whose amount significantly depends on the schedule type. When the number of scheduled streams exceeds 120, slack increases more rapidly. An increasing number of frames results in shorter time intervals between frame transmissions. Time intervals shorter than a guard band cannot be released for non-scheduled traffic and become slack instead.

We finally consider the utilization of bandwidth reserved for scheduled traffic in Figure 5(b). It slightly decreases with more than 100 streams, as then slack increases more than linearly. However, the difference among the schedule types remains significant regardless of the number of streams.

5) Impact of Frame Size: We vary the average frame size by adapting the maximum frame size of the streams. This experiment is relevant as many control messages in factory automation are small, i.e., average frame sizes are likely to be small. Table 2 compiles bandwidth usage for 100 scheduled streams. We observe that the bandwidth needed for guard bands slightly decreases for smaller frames and that more bandwidth can be provided for non-scheduled traffic. However, slack remains about constant, i.e., slack depends mostly on the number of frames but not on their size. As a result, the utilization of bandwidth reserved for scheduled traffic decreases with smaller frame sizes, which holds for all schedule types. However, this utilization suffers more for NRS and ERS than for TS. For very small frames, only 24.2% and 37.2% of the bandwidth reserved for scheduled traffic is utilized with NRS and ERS, respectively, the rest is slack.

6) Impact of Ring Size: We study the impact of the ring size on bandwidth usage by NRS, ERS, and TS. Table 2 compiles bandwidth usage depending on the ring size. The bandwidth

	NRS				ERS				TS			
Max. frame size (B)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)
1542	8.35	73.7	6.86	61.76	8.37	76.9	3.7	74.95	7.89	79.4	1.6	87.4
771	7.92	78.6	7.19	46.88	7.97	81.7	3.97	61.54	7.49	84.5	1.66	79.31
385	7.48	81.7	7.05	34.81	7.42	85	3.82	49.64	6.92	87.9	1.37	73.29
154	7.15	83.7	6.94	24.23	7.06	87	3.75	37.17	6.35	90.3	1.16	65.64
Ring size	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)
5	7.32	76.7	4.82	69.8	7.15	78.5	3.23	77.54	6.74	80.9	1.22	90.12
10	8.35	73.7	6.86	61.76	8.37	76.9	3.7	74.95	7.89	79.4	1.6	87.4
15	8.83	72.5	7.88	57.69	8.92	76.5	3.83	73.73	8.39	79.2	1.67	86.57
20	10.2	68.9	9.91	52.46	10.8	73.9	4.4	71.29	10.2	77	1.92	85.1
Proc. jitter (μs)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)
0.15	8.4	75.7	5	68.64	8.32	77.5	3.24	77.15	7.89	79.4	1.6	87.4
0.3	8.35	73.7	6.86	61.76	8.37	76.9	3.7	74.95	7.89	79.4	1.6	87.4
0.45	8.58	71.8	8.7	55.71	8.67	76.2	4.22	72.16	7.89	79.4	1.6	87.4
0.6	8.59	69.9	10.6	50.86	8.96	75.5	4.59	70.44	7.89	79.4	1.6	87.4
Sync. error (μs)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)	GBs (%)	Non-sched. traffic (%)	Slack (%)	ρ (%)
0.5	8.6	74.6	5.95	64.67	8.71	77.6	2.81	79.49	7.89	79.4	1.6	87.4
1	8.35	73.7	6.86	61.76	8.37	76.9	3.7	74.95	7.89	79.4	1.6	87.4
1.5	8.7	72.8	7.59	58.93	8.73	75.9	4.48	70.86	7.89	79.4	1.6	87.4
2	8.77	72	8.37	56.56	8.76	75	5.33	67.16	7.89	79.4	1.6	87.4

Tab. 2: Average bandwidth usage and schedule density per egress port for different maximum frame sizes, ring sizes, processing jitter, and clock synchronization error.

for guard bands increases with ring size for all schedule types in a similar manner. This is due to more dispersed traffic in larger rings, which is more difficult to group together for transmissions. We observe that slack increases with ring size, and so bandwidth for non-scheduled traffic decreases. Larger rings lead to longer paths and more uncertainty about a frame's arrival. Therefore, gaps between frames need to be larger, which increases slack. The performance of NRS suffers much more from larger rings than the one of ERS. The inefficiency of NRS results from the fact that its minimum gaps between frames strongly increase with the network size (cf. Equation 13) and heavily overestimate the necessary gap sizes.

7) Impact of Processing Jitter: Increasing processing jitter increases slack for both NRS and ERS. However, slack increases much more for NRS than with ERS. This is again due to the fact that NRS heavily overestimate needed gaps between frames. As a result, the bandwidth for non-scheduled traffic is more reduced for NRS than for ERS. TS does not respect any processing jitter.

8) Impact of Clock Synchronization Error: According to Table 2, slack increases with clock synchronization error about linearly. It is approximately the same growth for NRS and ERS. The reason is that synchronization errors are handled by NRS and ERS in the same way. Thus, the main advantage of ERS is its more efficient handling of processing jitter. TS does not respect any synchronization errors.

C. Admissible Streams

In the previous experiments, the number of streams was small, so all streams could successfully be scheduled. We now study the maximum number of admissible streams for the three scheduling approaches. That is, we increment a set of random streams until the ILP cannot find a valid schedule due to missing transmission resources in the network. The largest number of admissible streams in that set is one data point.

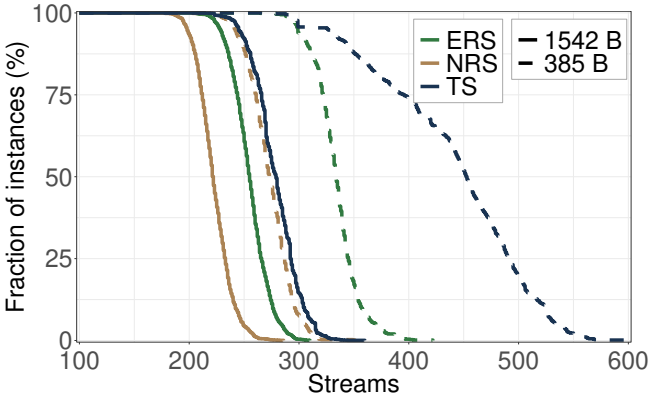


Fig. 6: Complementary cumulative distribution function (CCDF) of admitted streams based on 2000 problem instances for NRS, ERS, and TS. Values are reported for 1542 B and 385 B maximum frame size.

Figure 6 shows the complementary cumulative distribution functions (CCDF) of admissible streams for NRS, ERS, and TS. Each curve consists of 2000 data points. The solid lines correspond to large frames (1542 B max. frame size) and the dashed lines correspond to small frames (385 B max. frame

size). We observe that the number of admissible streams for a given schedule type varies a lot, which is mostly due to the properties of the individual streams (large or short periods, small or large frames) in the random set.

Max. frame size (B)	NRS	ERS/NRS	ERS	ERS/TS	TS
1542	222.6	1.146	255.1	0.9150	278.8
385	273.4	1.222	334	0.7555	442.1

Tab. 3: Average of admitted streams for different schedule types and maximum frame sizes as well as relative numbers.

The average number of admitted streams is compiled in Table 3. With ERS 14.6% more streams can be admitted than with NRS for large frames and 22.2% more for small frames. This is a very concrete benefit of ERS vs. NRS. It is in line with the observations that bandwidth reserved for scheduled streams can be more efficiently utilized by ERS than by NRS. In contrast, with ERS 8.5% less streams can be admitted than with TS for large frames and 24.5% less streams for small frames. This can be considered as the price of robustness.

D. Scalability

TAS scheduling is known to be NP-complete [8]. Nevertheless, most published scheduling algorithms in this area follow exact approaches. Thus, computation time and sizes of feasible problem instances matter. Therefore, most publications of TAS scheduling algorithms do not consider resulting schedule properties but focus on computation time, e.g., [4], [22], or [5]. Therefore, we investigate the scalability of computing ERS and compare it with four other well-known algorithms mentioned in Section IV.

The predominant approach for TAS scheduling in the literature uses SMT solving. In fact, there is no other ILP approach featuring GCL synthesis or robustness against non-determinism. Therefore, the scalability analysis gives also insights whether the use of ILP solving instead of SMT solving is a viable option for robust TAS scheduling.

The first algorithm (A1) is the SMT model by Craciunas et al. [4] which partially tackles time synchronization errors (see Section V) but cannot limit the number of used GCL entries. The second approach (A2) is from Santos et al. [10] which utilizes only a single gate closing and opening event per egress port during an entire hyperperiod. We use the models from Oliver et al. [5] (A3) and Jin et al. [19] (A4) because they feature real GCL synthesis and handle partially time synchronization errors. We omit the discussion about which properties of a problem instance influence scalability in general as such evaluations were carried out multiple times in the literature, e.g., in [4],[5],[22],[19].

1) Methodology: We evaluate the impact of the number of streams $n_{streams}$, the number of bridges in a network $n_{bridges}$, and the network topology τ. Ring, line, and star topologies are common topologies in traditional Ethernet networks. The line topology can be considered as a worst case scenario as frame transmissions compete for transmission times on a small set of links. The full-mesh topology is used as a theoretical best case counterpart for the line topology as every link carries a minimum number of frames, which results in least interference between frame transmissions. Scale-free networks are a class of random graphs similar to computer networks [25][26]. They feature many nodes with small node degree and a few highly connected hub nodes. The default parameters are $n_{streams} = 100$, $n_{bridges} = 10, \tau =$ Ring. We vary only one parameter at a time per experiment series. We construct 20 problem instances for each configuration of $n_{streams}, n_{bridges}$, and τ. Every problem instance is solved with all five methods, i.e., ERS proposed in this paper and the SMT-based algorithms A1 – A4.

We implemented model generation for A1, A3, and A4 while the model generation for A2 was obtained from [27]. We employ the SMT solver Z3 [28] to solve the models. The SMT solver is configured to use 4 parallel threads running the CDCL algorithm [29]. For a fair comparison, we also limit the number of parallel threads for the ILP solver to 4. As the runtime of A3 and A4 grows exponentially with the number of GCL entries per egress port, we limit this number to 8. We use a timeout of 1 h per problem instance. We report average solving times without model generation per parameter configuration and per approach.

# streams	ERS	A1 [4]	A2 [10]	A3 [5]	A4 [19]
10	0.0136	0.0523	5.16	3.32	3600
20	0.027	0.15	28.6	23.7	3600
50	0.152	0.963	414	598	3600
100	0.902	8.57	647	3600	3600
200	24	438	1204	3600	3600
# bridges	ERS	A1 [4]	A2 [10]	A3 [5]	A4 [19]
5	0.537	5.1	527	3600	3600
10	0.902	8.57	647	3600	3600
15	1.26	14.1	752	3600	3600
20	1.87	23	903	3600	3600
Topology	ERS	A1 [4]	A2 [10]	A3 [5]	A4 [19]
Ring	0.902	8.57	647	3600	3600
Line	3.35	28.1	1461	3600	3600
Star	0.445	3.73	118	3186	3600
Full-mesh	0.109	1.56	6.04	316	3600
Scale-free	0.627	4.94	409	3022	3600

Tab. 4: Computation times in seconds for different TSN scheduling algorithms. Default values are $n_{streams} = 100$, $n_{bridges} = 10$, and topology $\tau =$Ring. A timeout of 1 h per problem instance is applied, i.e., 3600 indicates that the computation did not complete.

2) Results: Table 4 compiles the average solving times for all five algorithms in seconds. ERS, A1, and A2 were able to schedule the problem instances of all experiments with ERS being the fastest, followed by A1 and A2. While A3 is faster than A2 for 10 and 20 streams, it cannot solve problem instances with 100 and more streams in the ring. A2 [10] outperformed A3 [5] for more than 50 streams. A3 [5] solved all problem instances with up to 50 streams in the ring topology, but was not able to schedule a single instance with

100 streams. A4 was not able to schedule a single instance in any of the experiments. We observe a fast growth of solving times for all algorithms when the number of streams increases. The reason for this is that model size grows quadratically with the number of frames transmitted over an egress port in all five approaches. This results in exponential worst case runtimes in the number of frames as scheduling for the TAS is NP-complete [8][30]. For the same reason, topologies with a few links, e.g., ring and line, are harder to schedule than highly meshed topologies, which can also be observed in the table. In contrast, solving times grow approximately linearly for increasing numbers of bridges in the ring topology.

For 100 streams or more, ERS outperforms all algorithms with regard to solving times roughly by a factor of 10, and those algorithms limiting used GCL entries by a factor of 50 or more. We explain this observation. A1 is an incremental approach, i.e., streams are scheduled on after another. This results in conflicting stream schedules and backtracking, which leads to longer solving times than ERS. The models of A3 and A4 use a binary decision variable for every frame and time interval with an open gate, i.e., a binary variable indicating whether a frame is transmitted in the respective time interval. Therefore, the solution space grows exponentially with the number of frames and with the number of GCL entries per egress port. Additionally, A4 divides time into discrete units and uses a binary variable for every time unit. This results in large models for problem instances with realistic stream periods.

In contrast, ERS uses only a single binary variable per frame per egress port to indicate whether the gate was closed in the time interval since the previously transmitted frame (see Section VII-B2). The resulting smaller solution space explains the vast difference in solving time to A3 and A4.

We conclude that the proposed modelling for ERS scales better compared to other known approaches for realistic sizes of problem instances. Models from literature (A3 [5] and A4 [19]) which feature non-trivial GCL synthesis failed even on medium-size instances. Thus, this evaluation shows that ILP-based approaches for robust TAS scheduling can be competitive to SMT-based approaches if the problem is modelled appropriately. Moreover, the modeling approach for ERS in this work may serve as a computation-efficient base for future works in this area that model additional constraints.

IX. Conclusion

In TSN periodic streams may be scheduled and the Time-Aware Shaper (TAS) may be used to protect scheduled streams from non-scheduled traffic. Schedules indicate transmission times at talkers, and gate openings and closings in bridges. Execution of such schedules may be affected by various sources of non-determinism: processing delays in bridges, clock synchronization errors in any device, and race condition from simultaneously arriving frames with same egress ports. Furthermore, schedules must respect the limited number of entries in the gate control list (GCLs) of the TAS. Although these issues are crucial for schedules working on real hardware, they have hardly been considered by existing scheduling algorithms.

This paper developed an ILP-based method for the computation of efficient robust schedules (ERS) that cope with the mentioned challenges. As an objective function, the remaining bandwidth for non-scheduled traffic is maximized, which is also novel in literature. To achieve robustness, ERS essentially leave sufficient but not constant time between consecutive frame arrivals, and between frame transmissions and consecutive gate closings. A subset of the ILP's constraints leads to tight schedules (TS) that do not provide extra time between frames. They resemble most schedules from literature. Another subset leads to naïve robust schedules (NRS) where constant time between frames and before gate closings is used to achieve robustness. This corresponds to a repair of some existing scheduling mechanisms coping with synchronization errors, plus their augmentation to cope with processing jitter, which has not been studied so far.

We evaluated and compared the different schedule types. We analyzed bandwidth usage and showed that ERS utilize bandwidth reserved for scheduled traffic more efficiently than NRS. We studied this issue depending on many parameters. The main advantage of ERS over NRS is that it better copes with processing jitter, which is due to the new modelling approach. TS lack robustness but can admit most scheduled streams, followed by ERS and NRS. With ERS 8.5% less streams could be admitted in our experiments than with TS, which is the price for robust schedules. With ERS, 14.6% more streams were admissible than with NRS, which demonstrates the efficiency of ERS. These numbers increase with smaller frames, larger networks, and more considered processing jitter and synchronization errors. Computing ERS is well feasible compared to well-known computation methods from the literature. The scheduling algorithm is significantly faster and can solve larger problem instances.

The computation method for ERS explicitly supports multicast streams. Future works may investigate the impact of multicast streams on admissible traffic. Further studies may extend ERS by the integration of protection mechanisms for use in safety-critical systems. Finally, the presented approach may facilitate the integration of streams coming from other networks or legacy devices. These frames may be subject to large jitter and the presented approach is able to construct schedules which take large jitter into account.

References

[1] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, pp. 1–1993, 2018.

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

[3] T. Stüber, L. Osswald, S. Lindner, and M. Menth, "A survey of scheduling algorithms for the time-aware shaper in time-sensitive networking (TSN)," *IEEE Access*, vol. 11, pp. 61 192–61 233, 2023.

[4] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," in *International Conference on Real-Time Networks and Systems (RTNS)*, 2016, pp. 183–192.

[5] R. S. Oliver, S. S. Craciunas, and W. Steiner, "IEEE 802.1Qbv Gate Control List Synthesis Using Array Theory Encoding," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2018.

[6] S. S. Craciunas and R. S. Oliver, "Out-of-Sync Schedule Robustness for Time-sensitive Networks," in *IEEE International Conference on Factory Communication Systems (WFCS)*, 2021, pp. 75–82.

[7] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pp. 1–421, 2020.

[8] F. Dürr and N. G. Nayak, "No-wait Packet Scheduling for IEEE Time-Sensitive Networks (TSN)," in *International Conference on Real-Time Networks and Systems (RTNS)*, 2016.

[9] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr, "Scaling TSN Scheduling for Factory Automation Networks," in *IEEE International Conference on Factory Communication Systems (WFCS)*, 2020.

[10] A. C. T. dos Santos, B. Schneider, and V. Nigam, "TSNSCHED: Automated Schedule Generation for Time Sensitive Networking," in *Formal Methods in Computer Aided Design (FMCAD)*, 2019, pp. 69–77.

[11] A. Varga, "OMNeT++," in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Güneş, and J. Gross, Eds. Springer Berlin Heidelberg, 2010, pp. 35–59.

[12] V. Gavriluţ and P. Pop, "Scheduling in Time Sensitive Networks (TSN) for Mixed-Criticality Industrial Applications," in *IEEE International Workshop on Factory Communication Systems (WFCS)*, 2018.

[13] Z. Feng, M. Cai, and Q. Deng, "An Efficient Pro-Active Fault-Tolerance Scheduling of IEEE 802.1Qbv Time-Sensitive Network," *IEEE Internet of Things Journal*, vol. 9, no. 16, pp. 14 501–14 510, 2021.

[14] M. Pahlevan and R. Obermaisser, "Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2018.

[15] M. Pahlevan, N. Tabassam, and R. Obermaisser, "Heuristic List Scheduler for Time Triggered Traffic in Time Sensitive Networks," *ACM SIGBED Review*, vol. 16, no. 1, pp. 15–20, 2019.

[16] W. Steiner, S. S. Craciunas, and R. S. Oliver, "Traffic Planning for Time-Sensitive Communication," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 42–47, 2018.

[17] M. Vlk, K. Brejchová, Z. Hanzálek, and S. Tang, "Large-scale Periodic Scheduling in Time-Sensitive Networks," *Computers & Operations Research*, vol. 137, 2022.

[18] M. Vlk, Z. Hanzálek, and S. Tang, "Constraint Programming Approaches to Joint Routing and Scheduling in Time-Sensitive Networks," *Computers & Industrial Engineering*, vol. 157, 2021.

[19] X. Jin, C. Xia, N. Guan, *et al.*, "Real-Time Scheduling of Massive Data in Time Sensitive Networks With a Limited Number of Schedule Entries," *IEEE Access*, vol. 8, pp. 6751–6767, 2020.

[20] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2021. [Online]. Available: https://www.gurobi.com.

[21] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl, "ILP-based Joint Routing and Scheduling for Time-Triggered Networks," in *International Conference on Real-Time Networks and Systems (RTNS)*, 2017.

[22] J. Falk, F. Dürr, and K. Rothermel, "Exploring Practical Limitations of Joint Routing and Scheduling for TSN with ILP," in *International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2018.

[23] J. Falk, F. Dürr, and K. Rothermel, "Time-Triggered Traffic Planning for Data Networks with Conflict Graphs," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.

[24] Industrial Internet Consortium, *Time Sensitive Networks for Flexible Manufacturing Testbed - Description of Converged Traffic Types*, [Online; accessed 19-September-2022], 2018. [Online]. Available: https://www.iiconsortium.org/pdf/IIC_TSN_Testbed_Traffic_Whitepaper_20180418.pdf.

[25] A.-L. Barabási, "Emergence of scaling in complex networks," in *Handbook of Graphs and Networks*. 2002, ch. 3, pp. 69–84.

[26] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '99, 1999.

[27] A. C. T. dos Santos, *TSNsched*, [Online; accessed 19-September-2022], 2019. [Online]. Available: https://github.com/ACassimiro/TSNsched.

[28] L. De Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *Theory and Practice of Software, International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[29] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem-Proving," *Communications of the ACM*, vol. 5, no. 7, pp. 394–397, 1962.

[30] W. Steiner, "An Evaluation of SMT-based Schedule Synthesis for Time-Triggered Multi-Hop Networks," in *IEEE Real-Time Systems Symposium (RTSS)*, 2010.

2.3 Scalability of Segment-Encoded Explicit Trees (SEETs) for Efficient Stateless Multicast

Segment-Encoded Explicit Trees (SEETs) for Stateless Multicast: P4-Based Implementation and Performance Study

Steffen Lindner*, Thomas Stüber*, Maximilian Bertsch*, Toerless Eckert†, and Michael Menth*
*University of Tuebingen, Chair of Communication Networks, 72076 Tuebingen, Germany
†Futurewei Technologies, CA 95050, United States

***Abstract*—IP multicast (IPMC) is used to efficiently distribute one-to-many traffic within networks. It requires per-group state in core nodes and results in large signaling overhead when multicast groups change. Bit Index Explicit Replication (BIER) and its traffic engineering variant BIER-TE have been introduced as a stateless transport mechanism for IPMC. However, they come with substantial operation, administration, and maintenance (OAM) costs and have scalability issues in large networks. In this paper, we present a novel stateless tree encoding mechanism called Segment-Encoded Explicit Tree (SEET). SEET encodes an explicit multicast distribution tree within a packet and is designed to be implementable on low-cost switching ASICs. Its design reduces OAM costs significantly and provides better scaling properties than BIER(-TE) in large networks. We implement SEET as a prototype for the Intel Tofino™ and present a simple, yet effective optimization heuristic that splits a SEET packet if its header becomes too large for modern switching ASICs. Evaluations highlight the advantages of SEET compared to BIER.**

***Index Terms*—Segment-Encoded Explicit Trees (SEETs), Bit Index Explicit Replication (BIER), multicast, IP networks, performance evaluation, optimization**

I. Introduction

IP multicast (IPMC) is the default multicast service in IP networks and is used to reduce the traffic load of one-to-many traffic. Examples for IPMC services are Multicast VPN, streaming, content delivery networks, or financial stock exchange [1]. Receivers, also called subscribers, of multicast services are organized in multicast groups that are identified by unique IP addresses. Traffic is forwarded along a multicast distribution tree to all subscribers of the multicast group. Thereby, only one packet is sent over each involved link. However, IPMC has two scalability issues. First, all forwarding nodes of the distribution tree need to maintain the forwarding state of the corresponding multicast groups. Second, when subscribers of a multicast group change, the forwarding nodes need to update their forwarding state, which results in excessive signaling overhead. Multicast mechanisms that rely on this kind of dynamic state are referred to as *stateful multicast* mechanisms. The Internet Engineering Task Force (IETF) is currently standardizing Bit Index Explicit Replication (BIER) [2] as a *stateless multicast* transport mechanism for IPMC traffic. BIER forwards IPMC traffic through a BIER domain without the need for dynamic forwarding state in core nodes. They leverage a so-called BIER bitstring that is added to the packets by ingress nodes of the domain. Each bit identifies an egress node of the BIER domain, i.e., a possible subscriber of the multicast group. If a bit is set, the corresponding egress node requires a packet copy. Based on this bitstring, core nodes of the BIER domain are able to forward the traffic to the appropriate egress nodes. However, when BIER is scaled to larger networks, multiple copies of a multicast packet might be forwarded over the same link, which mitigates the advantage of BIER over unicast forwarding. This is especially problematic for sparse multicast trees, i.e., multicast trees with a small number of receivers, in large networks. In that case, the ratio between receivers and redundant packets worsens.

The contributions of this paper are manifold. First, we show that BIER's scaling mechanism results in many redundant packet copies in large networks. Second, we present a novel mechanism for stateless multicast called Segment-Encoded Explicit Tree (SEET). It combines ideas of Segment Routing (SR) [3] and BIER [4], supports traffic engineering and has significant OAM advantages compared to BIER. Third, we present a P4-based implementation of SEET for the Intel Tofino™. SEET encodes the complete distribution tree within a packet header. If the required SEET header is too large to be processed by high-speed switching ASICs, multiple packets with a smaller SEET header are needed. We present a simple, yet effective heuristic that computes a near-optimal header fragmentation to split a large SEET header into multiple small headers. We compare the efficiency of SEET with BIER in different topologies.

The remainder of the paper is structured as follows. In Section II we describe related work. Then, we introduce BIER and show that BIER's scaling mechanism results in many redundant packet copies in large networks in Section III. Section IV introduces SEET and Section V gives a brief introduction to P4. Afterward, we present a P4-based implementation of SEET in Section VI. We present a simple, yet effective heuristic that determines how size-constrained near-optimal headers for SEET can be constructed in Section VII. We evaluate the scalability of BIER and SEET in different topologies in Section VIII and conclude the paper in Section IX.

II. Related Work

We first review related work for stateful multicast solutions. Then we discuss existing approaches for stateless multicast.

A. Stateful Multicast

IPMC is the default multicast service in IP networks. It was introduced in 1986 [5] and defines the transmission of IP datagrams to a set of hosts. Hosts dynamically join multicast groups, and the membership information of multicast groups is propagated through the network with the help of multicast routing protocols, e.g., PIM [6]. Islam et al. [7] and Al-Saeed et al. [8] provide a broad overview of stateful multicast services. They discuss shortcomings of IPMC regarding scalability and signaling overhead. Iyer et al. [9] present the Avalanche Routing Algorithm (AvRA) that leverages properties of data center topologies to compute optimized multicast distribution trees. They present an OpenFlow-based controller module that improves data rate by up to 12% and reduces packet loss by 51% compared to traditional IPMC. Dual-Structure Multicast (DuSM) [10] leverages the SDN paradigm to remove multicast management logic from switches. An SDN-based controller manages multicast group state on forwarding devices and balances traffic among multiple shared forwarding trees to avoid congestion. Further, the controller applies a multicast-to-unicast translation for multicast groups with low bandwidth to reduce the required state on forwarding devices. Voyer et al. [11] propose a new segment routing type for multi-point service delivery. The so-called SR replication segment instructs nodes to replicate packets to a set of downstream nodes in a SR domain. The mapping between a replication segment and a set of downstream nodes, called replication state, is held by the corresponding replication nodes. The replication state may change over time if the leaf nodes of a multi-point service change. Therefore, it resembles the dynamic forwarding state of traditional IPMC.

B. Stateless Multicast

Elmo [12] aims to improve the scalability of IPMC in data center environments. Multicast group information is encoded in the packet header, which eliminates the need for a dynamic state in forwarding devices. They claim to support up to one million different multicast groups in a three-tier data center topology with 27.000 hosts with an average packet header size of 114 bytes. Several works leverage Bloom filters to efficiently encode multicast traffic [13] [14] [15]. However, due to the inherent false-positive nature of Bloom filters, redundant or wrong forwarding decisions may be taken, which makes it unsuitable for a reliable multicast service. BIER [2] is currently standardized by IETF and proposes a novel stateless transport mechanism for IPMC. It is based on the notion of a bit string, in the following referred to as bitstring, that indicates the recipients of a multicast group. Forwarding devices within a BIER domain are able to forward BIER packets according to the bitstring without the need for dynamic state. Merling et al. [16] [17] and Lindner et al. [18] present a P4-based implementation of BIER on high-performance switching hardware. The presented prototypes are able to forward BIER-based multicast with 100 Gb/s per egress port. In subsequent work, Merling et al. [19] investigate the efficiency of BIER multicast in large networks. They compare the traffic savings of IPMC and BIER relative to unicast forwarding in a wide range of network topologies. Further, they present algorithms to build optimal BIER subdomains for large networks. BIER with tree engineering (BIER-TE) [20] augments BIER with traffic engineering capabilities. It is based on the same header format as BIER, i.e., a bitstring that indicates the recipients of a multicast group. Further, the bitstring contains a bit for each adjacency in the network. If the corresponding bit is set, the packet is forwarded over this adjacency. Hawkeye [21] enhances BIER-TE with a deep reinforcement learning agent that builds multicast distribution trees. Hawkeye can proactively compute multicast trees based on historical traces. MSR6 [22] implements BIER and BIER-TE based on the SRv6 [23] forwarding plane. It introduces a so-called RGB segment that contains the BIER bitstring and leverages unicast IPv6 forwarding between replication nodes. Eckert et al. [4] proposes Recursive BitString Structure (RBS) for BIER and MSR6 to improve the scalability for sparse multicast trees in large networks. They encode the forwarding tree in a hop-by-hop fashion using local bitstrings. Diab et al. [24] present YETI, a stateless and generalized multicast forwarding scheme. It is based on label and bitstring-based forwarding, similar to SEET, and can be used to encode an arbitrary multicast distribution tree. They compare it with existing rule-based multicast solutions as well as BIER-TE. However, they do not consider header limitations of modern forwarding ASICs and evaluate their solution only on small ISP backbone topologies with at most 197 routers and 486 links. In contrast, we consider realistic header limitations for SEET, propose an effective optimization heuristic that derives how a SEET header is fragmented into multiple packets, and evaluate SEET in large access network topologies with several thousand receivers.

III. Bit Index Explicit Replication (BIER)

We first give an overview of BIER(-TE) and explain its scaling mechanisms for large networks. Then we explain performance issues of BIER(-TE) in large networks with sparse multicast trees.

A. Overview

Bit Index Explicit Replication (BIER) [2] is a stateless transport mechanism for IPMC that has been standardized by the IETF. It is based on a domain concept and introduces three different types of BIER devices: Bit-Forwarding Ingress Routers (BFIRs), Bit-Forwarding Routers (BFRs), and Bit-Forwarding Egress Routers (BFERs). Figure 1 illustrates the concept of BIER.

Bit-Forwarding Ingress Routers (BFIRs) are the ingress nodes of the BIER domain. They receive IPMC packets ❶ and encapsulate them with a BIER header ❷. The BIER header contains a bit string, which we call BIER bitstring, that indicates the destinations of the packet within the domain. Each BFER is assigned to a bit position in the BIER bitstring. For simplicity, BFER n has been assigned to bit position[1] n in Figure 1. A bit is activated in the bitstring if the corresponding

[1]Bit position 1 corresponds to the lowest-significant bit.

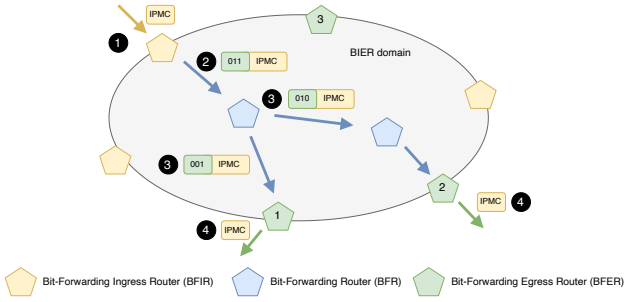


Fig. 1. A BIER domain is composed of Bit-Forwarding Ingress Routers (BFIRs), Bit-Forwarding Routers (BFRs), and Bit-Forwarding Egress Routers (BFERs).

BFER should receive a copy of the packet. Within the BIER domain, Bit-Forwarding Routers (BFRs) forward the BIER packet solely according to the BIER bitstring in the header. To that end, a BFR sends a packet copy to each next-hop over which at least one destination is reached. The bitstring is altered in each packet copy, such that it only contains the activated bits for BFERs that are reached via this next-hop ❸. This prevents duplicates at the receiver. Packets are forwarded according to the forwarding information from the routing underlay. Finally, Bit-Forwarding Egress Routers (BFERs) remove the BIER header and forward the IPMC packet as usual ❹.

BIER-TE augments the concept of BIER with tree engineering capabilities. It is based on the same header format as BIER, i.e., a bitstring that indicates the recipients of a multicast group. Further, the bitstring contains a bit for each adjacency in the network. If the corresponding bit is set, the packet is forwarded over this adjacency.

Both BIER and BIER(-TE) come with substantial OAM costs.

B. Scaling BIER(-TE) to Large Networks

The number of BFERs is limited by the size of the BIER bitstring. Common bitstring lengths are 256-, 512-, and 1024-bit. We refer to a BIER domain with bitstring length x as BIER-x. The bitstring length might be limited due to different reasons, e.g., technical restrictions in forwarding ASICs or header overhead tradeoffs. For example, a network with 10.000 receivers requires a 1250 bytes bitstring, which is not feasible in practice. BIER introduces subdomains to scale to larger networks. A BIER subdomain S_i is identified by the so-called Set Identifier (SI) in the BIER header. The SI remaps a bit position of the BIER bitstring to a different BFER for each subdomain, i.e., the first bit position identifies BFER 1 in S_1 and BFER 5600 in S_2. With this approach, a BIER domain can support 10.000 BFERs with a 256-bit BIER bitstring and 40 subdomains. Optimal SI selection, i.e., assigning a BFER to a SI in an optimal manner[2], is an NP-hard problem [19].

[2]An objective function might be to minimize the overall traffic rate or number of redundant packets.

Scaling BIER-TE to large networks is even more challenging as not only BFERs need to be part of a SI, but the whole distribution tree, i.e., all involved links. With limited header space, it is unclear how links should be assigned to SIs in a way such that traffic engineering is still viable. Furthermore, subdomains must be connected and need to contain backup paths in case of link or node failures.

C. Performance Issues

Large BIER domains may require multiple subdomains to reach all BFERs. If a BIER packet is destined to BFERs in several subdomains, multiple BIER packets are sent by the BFIR, one packet for each subdomain. Consequently, the same IPMC packet may be sent over a link multiple times. If the same IPMC packet is sent five times over the same link, four of these packets are redundant. This may reduce the advantage of BIER over native IPMC. We quantify the advantage of BIER over native IPMC with the following experiment. We consider a BIER domain with $n = 1024$ BFERs, an average node degree of eight, bitstring lengths of $b \in \{64, 128, 256, 512, 1024\}$ bits, and $s \in \{16, 8, 4, 2, 1\}$ subdomains, respectively. Then, we send BIER packets from a random source to n random receivers and repeat the experiment 50 times. We count the number of packets on all links p_{l_i} for both IPMC and BIER and report their difference, i.e., #redundant packets = $\sum_i p_{l_i}^{BIER} - \sum_i p_{l_i}^{IPMC}$. BFERs are randomly assigned to a single subdomain with equal probability. Figure 2 shows the number of redundant BIER packets that are sent to reach all receivers.

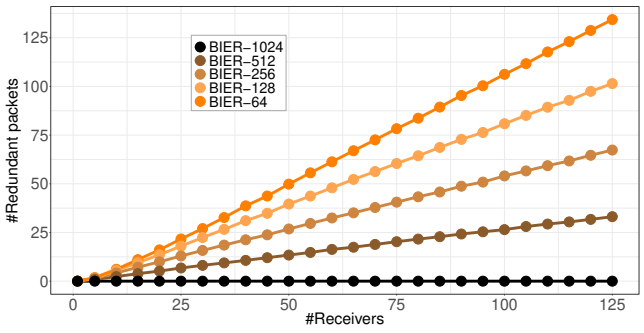


Fig. 2. Number of redundant BIER packets to reach all receivers.

Native IPMC and BIER-1024, i.e., BIER with a bitstring length of 1024 bit, send at most one packet over a link and do not require redundant packets to reach all destinations. BIER variants with smaller bitstring lengths require more redundant packets. The number of redundant packets scales with the number of receivers of an IPMC packet and is more severe in larger networks.

IV. Segment-Encoded Explicit Trees (SEETs)

In this section, we introduce a novel stateless tree encoding mechanism, which we call Segment-Encoded Explicit Trees (SEETs). It is based on ideas of Segment Routing (SR) and RBS/BIER but uses its own encoding for better efficiency.

First, we explain the general concept of a generic multicast tree encoding. Then we give an overview of SEET and explain its encoding in detail. Finally, we provide pseudocode for the forwarding logic of SEET-enabled devices.

A. Generic Multicast Tree Encoding

Stateless multicast solutions require the multicast distribution tree to be encoded within the packet. We propose the following generic encoding scheme that translates a recursive tree structure to a linear sequence of instructions that is agnostic to a specific protocol implementation. Figure 3 illustrates the generic encoding concept.

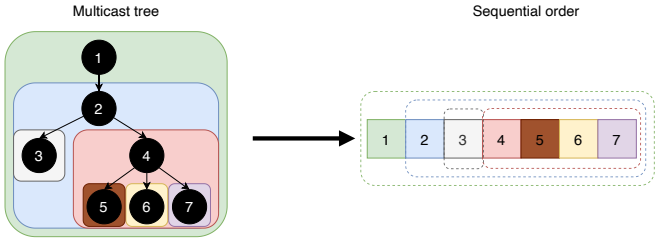


Fig. 3. Concept for stateless multicast source routing. A generic multicast distribution tree is translated into a sequential list structure.

A generic multicast distribution tree is encoded in a sequential list structure. Thereby, the tree structure is serialized into a list of elements. The forwarding principle of the stateless multicast source routing is illustrated in Figure 4. When a node receives an encoded multicast packet, it first partitions the encoded tree into its subtrees. Then, a packet copy is created for each next-hop. The packet copy contains only the relevant subtree, i.e., the subtree that starts with the next-hop. Thereby, the packet header shrinks along the forwarding path.

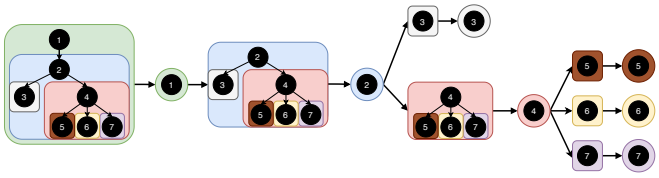


Fig. 4. Forwarding principle of the encoding concept for stateless multicast source routing. Only the relevant subtree of the packet header is forwarded to a downstream node in the multicast tree.

B. SEET Overview

SEET is a forwarding scheme to steer a multicast packet along an explicit or implicit multicast tree. It supports both shortest-path forwarding as well as traffic engineering. SEET's encoding scheme has been designed to be implementable on low-cost switching ASICs, e.g., with P4 [25] on the Intel Tofino™. A proof of concept implementation of SEET for the Intel Tofino™ is described in Section VI. Figure 5 illustrates the concept of SEET.

SEET is based on a domain concept similar to BIER and introduces three different types of devices: ingress nodes, forwarding nodes, and egress nodes. An ingress node receives

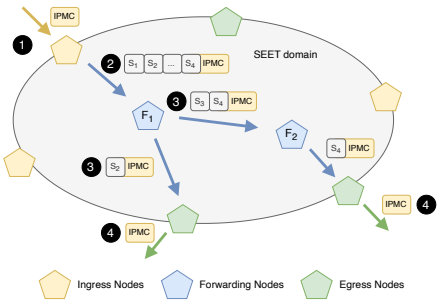


Fig. 5. A SEET domain is composed of ingress nodes, forwarding nodes, and egress nodes.

an IPMC packet and prepends a list of ordered segments to the packet ❶. We refer to this list of segments as forwarding stack (fs) ❷. Each segment encodes a SEET-specific identifier that is used by nodes to forward the packet along the distribution tree. Figure 5 illustrates how a SEET packet with four segments is received by the first forwarding node F_1. The initial segment S_1 identifies F_1 itself and instructs it to process the forwarding stack while the rest of the forwarding stack encodes the downstream multicast distribution tree. The first part of the forwarding stack, i.e., $\{S_2, ..., S_i\}$, encodes the downstream distribution tree for the first next-hop of F_1, and the second part of the forwarding stack, i.e., $\{S_{i+1}, ..., S_n\}$, encodes the downstream distribution tree for the second next-hop. When F_1 forwards the SEET packet to its neighbors, only the corresponding downstream forwarding stack is kept on the packet, the other part is removed. Finally, the egress nodes remove the SEET header and forward the underlying IPMC packet ❹.

C. SEET Encoding

The multicast distribution tree is recursively encoded in the forwarding stack. Figure 6 illustrates the encoding.

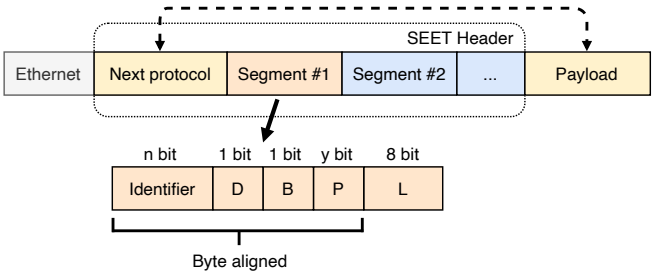


Fig. 6. A SEET header consists of a next protocol field and several segments that form the forwarding stack.

A SEET header consists of a 16 bit next protocol field and a list of segments, i.e., a forwarding stack (fs). The next protocol field is used to identify the protocol of the payload. A segment consists of a n-bit identifier, a deliver bit (D), a 1-bit bitstring indicator (B), y-bit padding (P), and an 8-bit length (L) field that indicates how many bytes are left in the current distribution tree. A byte alignment for segments is required to facilitate parsing in low-cost forwarding ASICs,

i.e., $(n + y + 1 + 1)\ mod\ 8 = 0$ has to hold. The identifier is used by forwarding nodes to determine the next-hop. For example, in a network with 100 nodes, seven bits suffice to identify all nodes in the network. The D-bit indicates whether the node that is identified through the identifier is a destination of the multicast tree.

D. Efficient Replication at Leaf Nodes

SEET requires for each recipient of the multicast distribution tree at least one segment[3]. Therefore, SEET can be efficiently used to encode paths that span multiple hops but it is less efficient if a node should replicate a packet to multiple neighbors. Therefore, we propose to use a BIER-like bitstring to address multiple neighbors efficiently at the penultimate hop in a multicast distribution tree. The B-bit indicates if a segment is followed by a BIER-like bitstring. In that case, the length field (L) is split into two 4-bit fields: bitstring length (BL) and bitstring set identifier (BSI). The first 4-bit indicate the length of the bitstring in bytes. The second 4-bit build an identifier with the same purpose as the SI in BIER. Bitstrings can only be used at the penultimate hop of a branch in the distribution tree. Figure 7 illustrates an example for the efficient replication at a leaf node. The example uses 14-bit global node identifiers and shows a distribution subtree where node #1052 is the penultimate hop with two receivers R_1 and R_3.

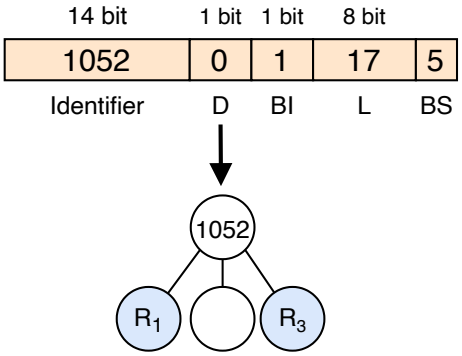


Fig. 7. The penultimate hop is addressed with a SEET identifier and carries a bitstring that is used to efficiently replicate the packet to multiple neighbors.

The identifier 1052 in the SEET segment addresses the penultimate hop. The D-bit is set to 0 as the node is not a receiver of the distribution tree. The B-bit is set to 1 as the segment is followed by a bitstring. Therefore, the length field with value 17 (`0b00010001`) is split into two 4 bit values, i.e., BL = 1 and BSI = 1. It is followed by a 1-byte long bitstring corresponding to the first SI. Finally, the bitstring `5 = 0b00000101` has the bits for receiver R_1 and R_3 set.

Additionally, if BL equals zero, the BSI can encode up to 16 (2^4) static multicast groups. For example, BL = 0 and BSI = 0 may be configured to trigger a local broadcast to a pre-defined set of neighbors.

[3]Multiple segments if an explicit path is encoded.

E. SEET Forwarding Algorithm

We formalize the above sketched forwarding algorithm using pseudocode. Algorithm 1 shows the forwarding logic for SEET in pseudocode for a packet p that has been received by a $node$ with forwarding stack $p.fs$. It uses the following methods without further formalization:

- $p.fs.pop()$: Removes the first segment in the forwarding stack fs of a packet p.
- $node.getNextHop(identifier)$: Returns the next-hop for a $node$ identified through the $identifier$.
- $p.fs.pop(l)$: Removes the first segment and the next l bytes in the forwarding stack fs of a packet p.

Algorithm 1: SEET forwarding algorithm.

```
Input: packet: p
       current node: node
1  if p.fs[0].identifier == node.identifier then
       /* Check if destination bit is set */
2      if p.fs[0].D then
3          copy packet to upper layer without SEET header;
4
       /* Check if BI bit is set          */
5      if p.fs[0].BI then
6          forward packet without SEET header according to bitstring;
7
8          return
9
10     p.fs.pop() ; /* Remove first segment */
11
12     while p.fs is not empty do
           /* next-hop of the packet      */
13         next_hop = node.getNextHop(p.fs[0].identifier) ;
14
           /* Keep relevant segments      */
15         create packet copy p_cp ;
16         p_cp.fs = next p_cp.l bytes ;
17
18         forward p_cp to next_hop ;
19
           /* Remove processed segments   */
20         p.fs.pop(p_cp.l) ;
21     end
22 end
23 else
       /* next-hop of the packet          */
24     next_hop = node.getNextHop(p.fs[0].identifier) ;
25
26     forward p to next_hop;
27 end
```

The first segment in the forwarding stack ($p.fs[0]$) identifies

the next-hop in the SEET domain. If a node receives a SEET packet, it first checks if the identifier of the first segment identifies the node itself (line 1). If the first segment does not identify the node, the packet is forwarded according to the identifier to the next-hop (line 26).If the first segment identifies the node itself, it is first checked if the destination bit (D-bit) is set (line 2). An activated D-bit indicates that the node is a destination in the multicast tree. In that case, the packet is copied and passed to the upper layer without the SEET header for native IPMC processing (line 3).

If the first segment has the B-bit set (line 5), then the node is a penultimate hop that uses a bitstring for efficient replication. In that case, the packet without SEET header is forwarded to all neighbors identified through the bitstring and the forwarding algorithm stops. If the B-bit is not set, the first segment is removed as it has been processed (line 10).

The following steps are performed as long as the forwarding stack is not empty. First, the next-hop of the packet is derived through the identifier of the first segment (line 13). Afterward, a packet copy is created that contains only the next $p_{cp}.l$ bytes of the forwarding stack (lines 15-16). Then, the packet copy is forwarded to the $next\_hop$ (line 18). Finally, the processed segments are removed from the original packet (line 20). The forwarding algorithm stops when all segments have been processed.

V. Introduction to P4

We review fundamentals of P4 that are relevant for the implementation of SEET. First, we give an overview of the general P4 pipeline of the Intel Tofino™. Then, we discuss the concept of recirculation and the capabilities of the packet parser. Details of the P4 language, its ecosystem, and related literature can be found in [26].

A. Overview

Programming protocol-independent packet processors (P4) [25] is a programming language used to describe the processing behavior of the data plane of compatible network devices, so-called targets. A P4 target follows an architecture that defines the processing pipeline and P4 primitives that are supported. Further, architectures can define so-called externs that extend the capabilities of the processing pipeline with target specific functions, e.g., support for cryptography. Figure 8 illustrates a simplified P4 pipeline of the Intel Tofino™ defined through the Tofino Native Architecture (TNA).

The processing pipeline of the Intel Tofino™ is divided into `ingress` processing, i.e., when a packet is received, and `egress` processing, i.e., when a packet is transmitted. When a packet is received, it is first parsed by the ingress parser, and relevant packet headers are extracted. Afterward, the received packet is processed according to the implemented processing logic in the ingress control. This may involve changing header fields, storing information in registers, and deciding which port the packet should be forwarded to. This is typically done by matching the previously extracted header fields against user-defined match+action tables (MATs). After ingress processing,

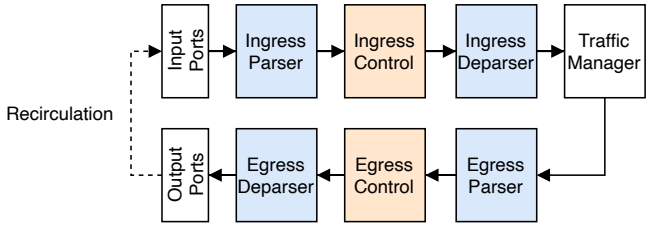


Fig. 8. Visualization of a simplified Tofino Native Architecture (TNA) [27] P4 pipeline. The pipeline consists of an ingress parser, ingress control, ingress deparser, traffic manager, egress parser, egress control, and egress deparser.

the packet is serialized through the ingress deparser and passed on to the traffic manager. The traffic manager is responsible for passing the packet to the correct egress port and performing packet replication, e.g., when a packet is cloned. Egress parser, egress control, and egress deparser have similar functionality as their ingress counterparts. After the egress deparser, the packet is physically transmitted through the corresponding egress port.

B. Packet Recirculation

P4 does not support the concept of loops. Therefore, iterative packet processing cannot be done within a single pipeline iteration. Iterative packet processing can be achieved through two mechanisms, called `resubmission` and `recirculation`. A resubmitted packet is immediately placed at the beginning of the ingress pipeline, i.e., at the ingress parser, after the initial ingress processing has finished. Therefore, resubmission allows to repeat the ingress processing on a packet. Resubmission can only be invoked during ingress processing. Further, the resubmitted packet corresponds to the initially received packet, i.e., all changes to the packet during the ingress processing are not applied. In contrast, recirculation takes place after egress processing. The packet is placed in the ingress parser as soon as the egress processing has stopped and all changes during the ingress and egress processing are applied. Recirculation on the Intel Tofino™ is a passive mechanism, which means that there is no P4 primitive that actively invokes recirculation. Ports can be configured to operate in a recirculation mode, i.e., packets transmitted through such a port are immediately placed in its ingress path again. This behavior is comparable to a physical loop. A packet that should be recirculated is forwarded through a port that operates in recirculation mode. We refer to such ports as recirculation ports.

C. Packet Parser

The packet parser extracts the relevant header fields used during ingress and egress processing. Both ingress and egress parsers are modeled as finite-state machine (FSM) and have a limited number of bytes that can be extracted depending on the capabilities of the ASIC. The parser divides the packet into extracted headers and its payload. The payload is not available during packet processing. Headers that are not extracted are considered to be part of the packet payload.

The parser extracts pre-defined headers according to the implemented FSM and the deparser emits previously extracted, possibly modified, (and still valid) headers. Figure 9 shows the definition of a custom header `example_header` and how it is extracted and emitted during parsing.

```
// header definition
header example_header {
    bit<8> type;
    bit<16> identifier;
}

// ingress parser
state parse_example_header {
    pkt.extract(hdr.example_header);
    // transit to the next state
    transition select(hdr.example_header.type) {
        ...
    }
}

// ingress deparser
// "add" previously extracted header to the packet
pkt.emit(hdr.example_header)
```

Fig. 9. Example of a custom header that is extracted in the ingress parser and emitted in the ingress deparser.

If a header is `invalidated` during ingress/egress processing, the header is not emitted in the deparser and, consequently, removed from the packet.

In addition to header extraction, P4 also supports to `advance` a packet during parsing. Thereby, the advanced bytes are removed from the packet. Figure 10 shows an example of a parsing state that advances the packet by 10 bytes.

```
// ingress parser
state remove_10_bytes {
    pkt.advance(8 * 10);
    // transit to the next state
}
```

Fig. 10. Bytes can be removed by advancing the packet.

Finally, the Intel Tofino™ provides a `ParserCounter` extern [27] that can be used to implement simple loops during parsing.

We leverage the capabilities of the `ParserCounter` extern, `extracting` (and keeping) header fields and `advancing` (and removing) bytes in our SEET implementation (see Section VI) to dynamically keep a certain number of segments and remove the remaining segments in a SEET header.

VI. P4 Implementation of SEET for Tofino

In this section, we give a brief overview of the P4 implementation of SEET for the Intel Tofino™. First, we give a high-level overview of the implementation. Then, we discuss its parsing logic in detail. The source code of SEET is available on GitHub[4].

[4]https://github.com/uni-tue-kn/seet

A. Overview

Most of the packet processing logic in general P4 pipelines is done within the so-called ingress and egress parts of the pipeline. However, with SEET, we need to be able to split the SEET header of a packet at an arbitrary byte position, which is not possible during regular ingress/egress processing. Therefore, most of the SEET processing logic is done within the parser. We leverage the capabilities of the parser to `extract` (and keep) header fields and to `advance` (and remove) bytes from a packet to remove parts of the SEET header dynamically during parsing. Figure 11 illustrates the concept of the implementation where a SEET packet should be split into two SEET packets.

When a SEET packet is received for the first time, it is parsed up to the first two segments[5]. Then, the packet is copied and both the original and the packet copy are equipped with a bridge header[6]. The bridge header contains two values K_{bytes} and R_{bytes}. The first value (K_{bytes}) specifies how many bytes should be extracted and kept from the SEET header. The second value (R_{bytes}) specifies how many bytes should be advanced and removed from the SEET header. Then, both packet versions are recirculated. After recirculation, both packets are parsed before they enter the ingress section of the pipeline. Thereby, the parser extracts the first K_{bytes} bytes from the SEET header and removes the next R_{bytes} bytes. In the example of Figure 11, a SEET header with length $L = X + Y$ should be split after X bytes. Therefore, the first packet copy extracts the first X bytes and removes the next Y bytes of the SEET header, and the second packet copy extracts zero bytes and removes the next X bytes. Finally, the first packet copy is forwarded to its intended neighbor, and the second packet is treated as a new SEET packet for further processing. This procedure is repeated until the whole SEET header has been processed[7].

If the first segment has the bitstring indicator (B) set, the contained IPMC packet is replicated according to the local bitstring to all relevant neighbors without SEET header. Forwarding logic for bitstring-based replication is similar to [17] [18]. If the first segment has the deliver bit (D) set, an additional packet copy is passed to the upper layers of the device.

B. Parsing Logic

Packets that are recirculated are received on special ports and always carry a bridge header that contains the number of bytes that should be extracted (K_{bytes}) and the number of bytes that should be removed (R_{bytes}). Thereby, K_{bytes} is split into two fields: `tens` and `units`.

`Tens` represents the number of 10 bytes that should be extracted, and `units` represents the number of single bytes

[5]If a bitstring follows the first segment, the bitstring is parsed instead.

[6]A bridge header is a header that is temporarily prepended to the Ethernet header for local processing. The header is removed when the packet is forwarded to its final destination.

[7]The length field of the original first segment is used to determine whether the whole SEET header has been processed.

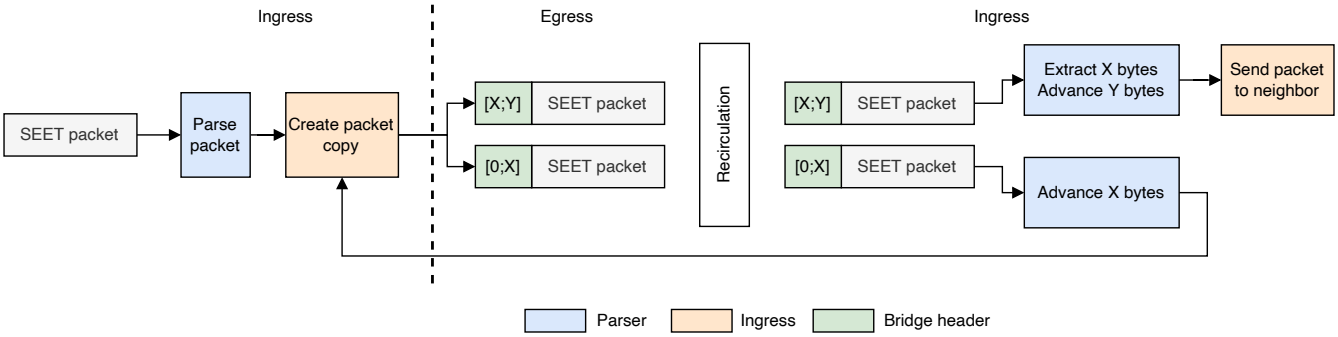


Fig. 11. High-level implementation overview of the SEET forwarding logic. Most of the processing logic is done within the parser.

that should be extracted. Therefore, if 85 bytes should be extracted, `tens` equals eight and `units` equals five. We defined seven different headers with sizes of {100, 50, 20, 10, 5, 2, 1} byte(s). The 20-byte and 2-byte headers are implemented as header stacks of size 2, i.e., up to two 20-byte and up to two 2-byte headers can be extracted within the same header stack. Further, we defined 14 different parsing states P_i^K, $i \in [1, 14]$ that combine the different headers to extract $i \cdot 10$ bytes depending on the value of `tens`. For example, the parsing state P_{13}^K extracts one 100-byte header, one 20-byte header, and one 10-byte header. Similarly, nine additional parsing states extract up to 9 bytes depending on the number of `units`. This approach can extract between 0 and 149 bytes with at most two parse state transitions.

Afterward, the `ParserCounter` extern is used to advance the packet R_{bytes} times by one byte. The combination of extracted headers and the `ParserCounter` extern ensures that the maximal parse depth is not exceeded.

VII. Fragmentation Algorithm

We introduced an encoding for Segment-Encoded Explicit Tree (SEET) in Section IV. It represents the forwarding tree of a packet in the packet's header. However, so far we ignored that the maximum header size that can be processed by forwarding nodes is limited due to technical restrictions. Therefore, multiple packets may be required to deliver a message to all of its receivers. We present a simple yet efficient algorithm to fragment a message into multiple packets such that traffic overhead is minimized. First, we give an overview of the idea of the algorithm. Then, we present its details. Afterwards, we discuss its runtime. Finally, we illustrate the algorithm by a brief example.

A. Overview

The presented encoding features two major ideas for minimizing the representation of a multicast tree. First, a long subpath of the path to a receiver can be bridged by a single SEET segment. Therefore, it is reasonable to address receivers that share long subpaths with the same packet. However, every replication in the multicast tree requires an additional SEET header. Thus, the number of replications in a multicast tree should be small. Second, multiple receivers with a common penultimate hop can efficiently be addressed by a local bitstring. We conclude that receivers should be grouped such that the multicast trees of the resulting packets contain few replication nodes except for replications at the penultimate hop.

B. The Algorithm

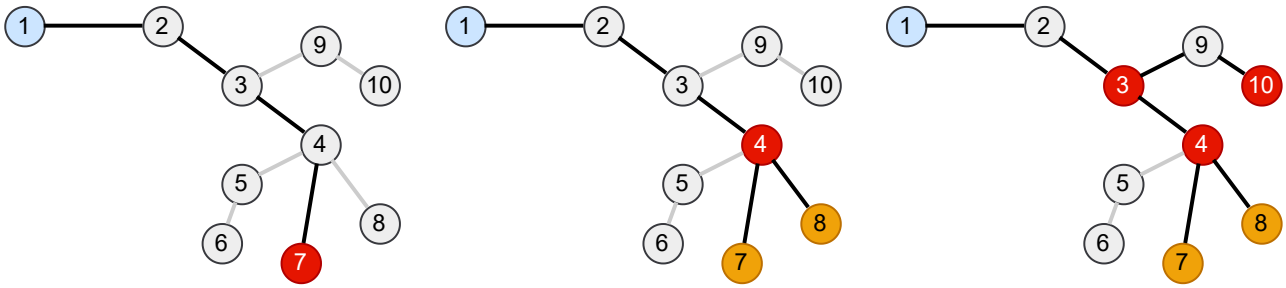
We propose a simple yet efficient algorithm which groups receivers according to the above observation. Given are a network topology, a source node, a set of receivers, and the desired paths for all source-receiver pairs. A forwarding tree is constructed by merging paths with common subpaths at the last node present in both paths. Likewise, a tree and a path are merged by adding a new branch to the tree at the last node present on the path. The algorithm starts with an empty packet header. A depth-first search in the forwarding tree is started at the source node of the message. Every time a receiver r of the message is discovered, it is added to the packet header according to exactly one of the following cases:

1) If the header is empty, a SEET header to r is introduced.
2) If the header does not contain a SEET header with a common subpath to r, a SEET header to r is introduced.
3) If the header contains a SEET header s to the penultimate hop of r, r is included in the local bitstring of s.
4) If a SEET header s addresses a node r' with the same penultimate hop p as r, s is removed from the header, a SEET header to p is introduced, and r and r' are added to the local bitstring of the new SEET header.
5) If the header contains some SEET header s with a common subpath to r and none of the other cases applies, a SEET header to the last possible replication node of s and r is inserted before s and a SEET header to r is introduced.

If the resulting packet header exceeds the maximum header length, the discovered receiver is not added to the header. Instead, the current packet is finished and the receiver is added to a new packet header. The algorithm terminates when all receivers are added to a packet.

C. Runtime

Let V and E be the sets of vertices and edges in the network topology. Depth-first search has a runtime of $\mathcal{O}(|V| + |E|)$.



(a) After discovering node 7. (b) After discovering node 8. (c) After discovering node 10.

Fig. 12. Depiction of the algorithm for a message sent from node 1 to the nodes 7, 8, and 10. Red nodes are addressed by a SEET header while orange nodes are addressed by a local bitstring.

In the worst case, every node of the topology is a receiver. Deciding which of the above cases applies and finding the replication point in case 5) can be done in $\mathcal{O}(|E|)$ if the multicast tree of the current packet is stored as list of edges in topological order. Thus, the runtime of the presented algorithm is $\mathcal{O}(|V| \cdot |E|)$ in the worst case. Typically, this is an heavy overestimation as the multicast tree of the current packets contains significantly less than $|E|$ edges.

D. Illustrating Example

We explain the algorithm by a brief example. Figures 12(a)–12(c) depict a network topology and three steps of the algorithm. A message should be sent from node 1 to the nodes 7, 8, and 10. The nodes are discovered in ascending order. The headers after the steps of the example are shown in Figure 13. Initially, no receiver is covered by the packet's header. When node 7 is discovered in Figure 12(a), case 1) from the algorithm's description applies. Thus, a SEET header with node 7 as destination is introduced. Then, node 8 is discovered in Figure 12(b). The current header contains a SEET header to a node with the same penultimate hop as node 8. Thus, case 4) of the algorithm's description applies. The SEET header to node 7 is removed and a SEET header to the penultimate hop, node 4, is introduced instead. Nodes 7 and 8 are addressed by the local bitstring of node 4. Finally, node 10 is discovered in Figure 12(c). The path to node 10 shares a common subpath with the path of the already existing SEET header to node 4. Thus, case 5) of the algorithm's description applies. The latest possible replication node to reach node 4 and node 10 is node 3. Thus, a SEET header to node 3 is introduced which contains SEET headers to node 4 and node 10 recursively.

Eventually the header will exceed the size limit in larger topologies with more receivers than in the presented example. The header is considered full in this case and a new empty header is the new working header. The depth-first search proceeds with the last discovered node and the algorithm terminates when all receivers were discovered.

VIII. Evaluation

We evaluate the encoding and the message fragmentation algorithm. To that end, we compare the presented approach with traditional IPMC and BIER. First, we introduce the methodology of the evaluations. Then, we motivate the fragmentation algorithm by evaluating the header sizes imposed by SEET. Afterward, we evaluate the relative overhead of SEET and BIER with respect to packet transmissions compared to IPMC. Finally, we present results regarding the overall traffic transmitted in the network.

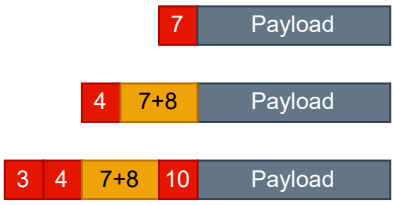


Fig. 13. Headers aftet discovering nodes 7, 8, and 10 in the example. SEET headers are depicted red while local bitstrings are depicted orange. Numbers indicate the destinations of the respective header.

A. Methodology

We give the details of the evaluation setup such as network topology, traffic model, and evaluation metrics.

1) Network Topology: We sampled 20 graphs with 1024 nodes according to the Waxman model [28] such that the average node degree is 4. The nodes of these graphs represent the core nodes of a distribution network. For each core node, we added 16 end systems and connected them to the respective core node. Thus, the resulting network topologies contain $(16 + 1) \cdot 1024 = 17408$ nodes and $(16 + 2) \cdot 1024 = 18432$ links.

2) Traffic Model: For every network topology n and number of receivers $r \in \{1, 2, 4, ..., 16384\}$, we sampled 20 sets of r receivers from the set of end systems of n. For every such set of receivers $\mathcal{R}$, a message is send from every end system to all end systems in $\mathcal{R}$. The same sets are used to evaluate SEET, BIER, and IPMC. We remark that randomized sets of receivers constitute a worst case for SEET as the local bitstrings cannot be leveraged for leafs of different core nodes.

3) Metrics: We calculate results with three metrics: maximum header size, relative packets, and relative traffic.

a) Maximum header size: The maximum header size is the number of bytes required to encode a set of receivers, excluding headers of lower layers and IP headers.

b) Source packets: The source packets metric captures the load imposed to source nodes due to packet construction. It is the number of packets sent per source node averaged over all end systems. We remark that IPMC requires exactly one source packet regardless the the set of receivers. Thus, all results can be considered to be relative to IPMC.

c) Relative packets: The relative packets metric represents the overhead of individual packet hops compared to IPMC. Let p_{IPMC} be the number of packet hops required to sent a message from some source node to some set of receivers via IPMC. If an alternative multicast approach $A \in \{\text{BIER}, \text{SEET}\}$ requires p_A packet transmissions for the same source node and set of receivers, the relative packets metric is formally defined as $\frac{p_A}{p_{\text{IPMC}}}$. For every number of receivers r, we report results averaged over all source nodes, sets of receivers with size r, and network topologies for the maximum header size and the relative packets metrics.

d) Relative traffic: The relative traffic metric captures the overhead of data transmitted in the network for a given set of receivers. Thus, it is the sum of the sizes of all packet hops for all source end systems, including payload and IP headers, relative to IPMC. We assume a payload of 500 B as empirical studies suggest this is the average payload of IP packets in the Internet [29]. For every number of receivers r, we report results averaged over all network topologies and receiver sets of size r.

B. Header Size

The header size of IPMC and BIER packets is predefined and does not depend on the set of receivers. This is not the case for SEET due to its tree engineering capabilities. Figure 14 depicts the average initial header size resulting from sending a message to varying numbers of receivers.

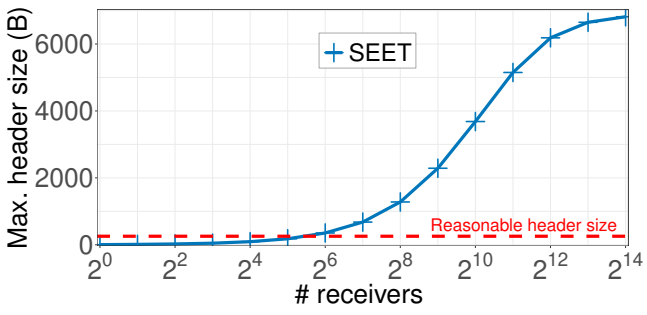


Fig. 14. Average initial header size for varying numbers of receivers. The dashed line indicates the maximum header size that can be processed by the presented implementation.

We observe a fast increase of the header size for moderate numbers of receivers ($r \geq 2^6$). With an increasing number of receivers, the header size reaches a plateau eventually. The reason for this behavior is the following. If two receivers are not leave of the same core node by chance, separate SEET headers are required to reach them. With an increasing number of receivers, chances are high that every core node is already included in the header. Thus, additional receivers can be added by simply flipping the corresponding bits in the local bitstrings of the SEET headers without increasing its sizes.

The dashed line indicates the maximum header size that can be processed by reasonable forwarding hardware. With such a header limit, only ~ 32 receivers can be addressed by a single packet in the case of uncorrelated receivers. We conclude that the message fragmentation algorithm from Section VII is necessary under realistic conditions. In contrast, BIER can encode 2048 receivers with the same header limit. Therefore, SEET is less efficient with respect to encoding denseness. However, SEET headers decrease in length on their path. Thus, no conclusions regarding the overall traffic volume can be drawn.

C. Source Packets

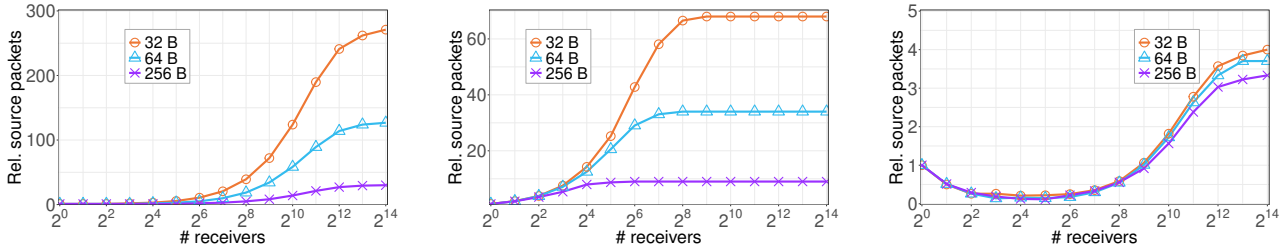
We regard header sizes of more than 256 MB as infeasible for practical applications due to hardware restrictions ins forwarding devices. Thus, packets with more than 2^6 receivers must be split into multiple packets. However, the fragmentation of receivers into subsets matters with respect to the metrics from Section VIII-A3. We used the fragmentation algorithm of Section VII for this purpose. The fragmentation of receivers into subsets results in multiple packets per message send from an end system which imposes additional overhead.

Figures 16(a)–16(c) depict the average number of packets sent per end system. We observe that the number of source packets increases for larger sets of receivers (Figure 16(a)). This is consistent with the results from Section VIII-B. In the case of BIER (Figure 16(b)) the number of source packets saturates for rather small receiver sets and does not increase further. This is due to the design of BIER as only a single source packet per SD is required. Thus, the maximum number of source packets is sent when at least one receiver per SD is addressed.

Comparing SEET directly to BIER (Figure 16(c)), we see that SEET sends several times more packets than BIER in the case of many receivers. A BIER packets uses only a single bit in its header per receiver. While SEET also encodes some receivers with individual bits, replication nodes must be encoded with identifiers and subheaders. This in turn results in less receivers per header or more packets sent. However, in the case of small receiver subsets, SEET requires less source packets than BIER. This is due to BIER sending one packet per SD while SEET can address these receiver sets with a small number of packets.

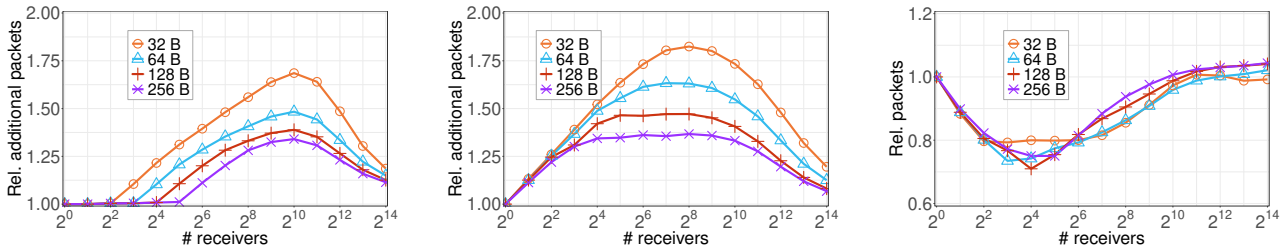
D. Packet Overhead

We showed that the SEET encoding is less efficient than BIER with respect to the number of receivers addressable with a single packet. However, the forwarding tree of a packet with



(a) SEET relative to IPMC. (b) BIER relative to IPMC. (c) SEET relative to BIER.

Fig. 15. Average number of source packets.



(a) SEET relative to IPMC. (b) BIER relative to IPMC. (c) SEET relative to BIER.

Fig. 16. Relative numbers of packet transmissions for varying numbers of receivers and different maximum header sizes.

a small number of receivers contains less hops. Additionally, BIER and SEET require multiple packets for large sets of receivers which results in redundant packet transmissions compared to IPMC. Thus, it is not clear whether the remarks regarding encoding efficiency translate to the number of packet transmissions.

Figures 16(a)–16(b) depict the relative packet overheads of BIER and SEET for different maximum header sizes compared to IPMC. We observe that SEET (Figure 16(a)) and BIER (Figure 16(b)) benefit from larger headers as more receivers can be encoded within a single packet. Consequently, less additional packets need to be sent.

Further, the relative packet overhead of BIER and SEET compared to IPMC decreases for large sets of receivers. If a core node is already receiving a BIER or a SEET packet, forwarding it to an additional leaf of this core node requires only a single hop. The same does hold for IPMC which implies that the relative packet numbers decline.

Figure 16(c) compares SEET directly to BIER. We see that SEET requires less or an equal number of packet transmissions than BIER. At first this result seems counterintuitive as BIER can address more receivers with a single packet. However, BIER subdomains are statically configured and may be sub-optimal from the perspective of some source nodes. In case of rather small sets of receivers BIER requires an individual packet per subdomain that contains at least one receiver. In contrast, SEET packets are individually optimized for every source node and set of receivers. Thus, a single packet is sufficient in many cases.

E. Traffic Overhead

The number of packets is an important metric for the processing load of forwarding hardware. Switching ASICs are limited by the number of packets that can be processed per second. However, network congestion and quality of service depend on the overall amount of traffic that must be transmitted. The total traffic amount depends on the number of individual packet hops and the size of a packet. Thus, there is a non-trivial tradeoff between reducing the number of packet hops or the header size. We compare SEET and BIER with respect to this tradeoff.

Figure 17 shows the traffic overhead of BIER and SEET relative to IPMC with small and large headers.

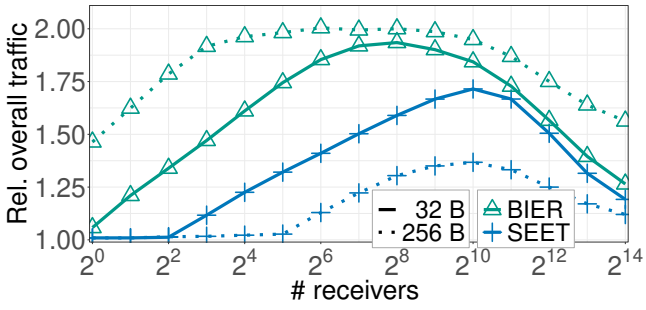


Fig. 17. Relative traffic of BIER and SEET for varying numbers of receivers and different maximum header sizes.

First, we observe a similar trend as in Figures 16(a)–16(b). With an increasing number of receivers, the relative overall

traffic increases until almost all core nodes are already part of the distribution tree. Then, additional receivers can be addressed by simply flipping a bit in a local bitstring. Further, we observe that SEET results in less traffic overhead than BIER. While BIER is more efficient in encoding receivers into a packet's header, the size of a BIER header does not change along the packet's path. In addition, many bits of the header bits are set to 0, even for large sets of receivers. The topology under consideration consists of 16384 end systems. Thus, even with 8192 receivers, half of the header space is not efficiently used and only filled with zeros.

In contrast, these drawbacks of BIER do not apply for SEET. SEET can leverage the whole header limit at every packet to encode the distribution tree. Further, the size of the SEET header reduces at every replication node, and only the relevant parts are relayed to the respective subtree. Nodes that are not receivers of a packet are not represented in the header.

F. Performance

We measure the number of multicast groups that can be fragmented into packet headers per second. The measurement was performed on an AMD EPYC 7543 @ 2.8 GHz with 32 cores. The machine is equipped with 128 GB of RAM. However, the computation requires only 110 MB of RAM for all 32 threads combined. The algorithms and the evaluations were programmed in Rust. Figure 18 depicts the number of multicast groups that are processed per second. Even in the case of 16384 receivers, the presented approach constructed packets for more than 1000 multicast groups per second. Overall, runtimes increase linearly with the number of receivers in the evaluation scenario. Thus, we conclude that SEET and the fragmentation algorithm are suitable for applications with high multicast turnover rates.

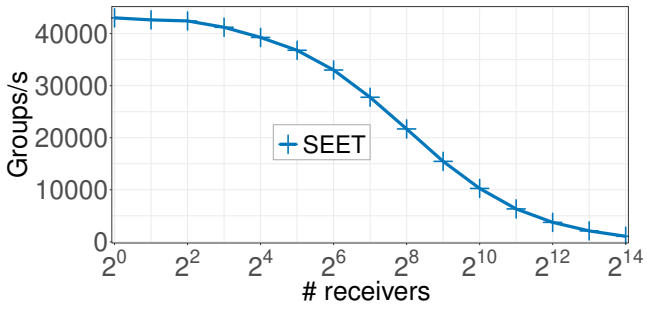


Fig. 18. Number of multicast groups that can be fragmented into packet headers per second.

IX. Conclusion

In this work, we proposed a novel stateless multicast protocol denoted as Segmented-Encoded Explicit Trees (SEETs). SEET encodes the forwarding tree of a packet in the packet's header. Thus, SEET is an enabler for tree engineering. However, constructing a SEET packet is complex. The encoding of large forwarding trees results in large packet headers. We developed an algorithm to fragment the set of receivers of a packet such that the resulting subsets can addressed by a single packet, respectively. We employed this algorithm to compare BIER and SEET in a quantitative comparison study. The results showed that SEET results in less source packets, packet transmissions, and overall traffic compared to BIER. Thus, we conclude that SEET is a viable alternative for BIER. However, the comparison only featured tree-like hierarchical topologies with high fan-outs before leaf nodes. There is a variant of BIER which features tree-engineering denoted as BIER-TE. Unfortunately, the scalabiltiy of BIER-TE has not been evaluated so far and there is no partitioning algorithm for BIER-TE in the literature. Future works may propose such an algorithm and employ it to compare SEET with BIER-TE.

References

[1] N. K. Nainar, R. Asati, M. Chen, X. Xu, A. Dolganow, T. Przygienda, A. Gulko, D. Robinson, V. Arya, and C. Bestler, "BIER Use Cases," Internet Engineering Task Force, Internet-Draft, Sep. 2020, work in Progress. https://datatracker.ietf.org/doc/draft-ietf-bier-use-cases/12/.

[2] I. Wijnands, E. C. Rosen, A. Dolganow, T. Przygienda, and S. Aldrin, "RFC8279: Multicast Using Bit Index Explicit Replication (BIER)," Internet Engineering Task Force, Request for Comments, Nov. 2017, https://www.rfc-editor.org/info/rfc8279.

[3] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir, "Segment Routing Architecture," RFC 8402, Jul. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8402

[4] T. Eckert, M. Menth, X. Geng, X. Zheng, R. Meng, and F. Li, "Recursive BitString Structure (RBS) Addresses for BIER and MSR6," Internet Engineering Task Force, Internet-Draft draft-eckert-bier-rbs-00, Oct. 2022, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-eckert-bier-rbs/00/

[5] S. E. Deering, "Host extensions for IP multicasting," RFC 988, Jul. 1986. [Online]. Available: https://www.rfc-editor.org/info/rfc988

[6] B. Fenner, M. J. Handley, H. Holbrook, I. Kouvelas, R. Parekh, Z. J. Zhang, and L. Zheng, "Protocol Independent Multicast - Sparse Mode (PIM-SM): Protocol Specification (Revised)," RFC 7761, Mar. 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7761

[7] S. Islam, N. Muslim, and J. W. Atwood, "A Survey on Multicasting in Software-Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. 20, pp. 355–387, 2018.

[8] Z. AlSaeed, I. Ahmad, and I. Hussain, "Multicasting in Software Defined Networks: A Comprehensive Survey," *Journal of Network and Computer Applications*, vol. 104, pp. 61–77, 2018.

[9] A. Iyer, P. Kumar, and V. Mann, "Avalanche: Data Center Multicast using Software Defined Networking," in *International Conference on COMmunication Systems and NETworks (COMSNETS)*, 2014.

[10] W. Cui and C. Qian, "Scalable and Load-Balanced Data Center Multicast," in *IEEE Globecom*, 2015.

[11] D. Voyer, C. Filsfils, R. Parekh, H. Bidgoli, and Z. J. Zhang, "SR Replication segment for Multi-point Service Delivery," Internet Engineering Task Force, Internet-Draft draft-ietf-spring-sr-replication-segment-15, Jun. 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-spring-sr-replication-segment/15/

[12] M. Shahbaz, L. Suresh, J. Rexford, N. Feamster, O. Rottenstreich, and M. Hira, "Elmo: Source Routed Multicast for Public Clouds," in *ACM SIGCOMM*, 2019, p. 458–471.

[13] P. Jokela, A. Zahemszky, S. Arianfar, P. Nikander, and C. Esteve, "LIPSIN: Line speed Publish/Subscribe Inter-Networking," in *ACM SIGCOMM*, Barcelona, Spain, Aug. 2009.

[14] M. J. Reed, M. Al-Naday, N. Thomos, D. Trossen, G. Petropoulos, and S. Spirou, "Stateless Multicast Switching in Software Defined Networks," in *IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–7.

[15] Z. Chen, J. Huang, Q. Wang, J. Liu, Z. Li, S. Zhou, and Z. He, "MEB: an Efficient and Accurate Multicast using Bloom Filter with Customized Hash Function," in *Asia-Pacific Workshop on Networking*, 2023, pp. 157—-163.

[16] D. Merling, S. Lindner, and M. Menth, "P4-Based Implementation of BIER and BIER-FRR for Scalable and Resilient Multicast," *Journal of Network and Computer Applications*, vol. 169, Nov. 2020.

[17] ——, "Hardware-Based Evaluation of Scalable and Resilient Multicast With BIER in P4," *IEEE Access*, vol. 9, pp. 34 500–34 514, Feb. 2021.

[18] S. Lindner, D. Merling, and M. Menth, “Learning Multicast Patterns for Efficient BIER Forwarding with P4,” *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1238–1253, Jun. 2023.
[19] D. Merling, T. Stüber, and M. Menth, “Efficiency of BIER Multicast in Large Networks,” *IEEE Transactions on Network and Service Management*, 2023.
[20] T. Eckert, M. Menth, and G. Cauchie, “Tree Engineering for Bit Index Explicit Replication (BIER-TE),” RFC 9262, Oct. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9262
[21] L. Lu, Q. Li, D. Zhao, Y. Yang, Z. Luan, J. Zhou, Y. Jiang, and M. Xu, “Hawkeye: A Dynamic and Stateless Multicast Mechanism with Deep Reinforcement Learning,” in *IEEE Infocom*, May 2023.
[22] Y. Liu, J. Xie, X. Geng, and M. Chen, “RGB (Replication through Global Bitstring) Segment for Multicast Source Routing over IPv6,” Internet Engineering Task Force, Internet-Draft draft-lx-msr6-rgb-segment-04, Mar. 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-lx-msr6-rgb-segment/04/
[23] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li, “Segment Routing over IPv6 (SRv6) Network Programming,” RFC 8986, Feb. 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc8986
[24] K. Diab and M. Hefeeda, “Yeti: Stateless and Generalized Multicast Forwarding,” in *USENIX Syposium on Networked Systems Design & Implementation (NSDI)*, Apr. 2022, pp. 1093–1114.
[25] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, “P4: Programming Protocol-Independent Packet Processors,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, 2014.
[26] F. Hauser, M. Häberle, D. Merling, S. Lindner, V. Gurevich, F. Zeiger, R. Frank, and M. Menth, “A survey on data plane programming with P4: Fundamentals, advances, and applied research,” *Journal of Network and Computer Applications*, vol. 212, 2023.
[27] Intel, “P416 intel tofino native architecture - public version,” https://github.com/barefootnetworks/Open-Tofino, 2021.
[28] B. Waxman, “Routing of multipoint connections,” *IEEE Journal on Selected Areas in Communications*, vol. 6, pp. 1617–1622, 1988.
[29] F. Liu *et al.*, “The packet size distribution patterns of the typical internet applications,” in *IEEE International Conference on Network Infrastructure and Digital Content*, 2012, pp. 325–332.

3 Accepted Manuscripts (Additional Content)

3.1 Comparison of Forecasting Methods for Energy Demands in Single Family Homes

Comparison of Forecasting Methods for Energy Demands in Single Family Homes

Thomas Stüber, Ricarda Hogl, Bernd Thomas, Michael Menth
University of Tuebingen, Tuebingen, Germany, {thomas.stueber, ricarda.hogl, menth}@uni-tuebingen.de
Reutlingen University, Reutlingen, Germany, bernd.thomas@reutlingen-university.de

Abstract

The integration of renewable energy sources in single family homes is challenging. Advance knowledge of the demand of electrical energy, heat, and domestic hot water (DHW) is useful to schedule projectable devices like heat pumps. In this work, we consider demand time series for heat and DHW from 2018 for a single family home in Germany. We compare different forecasting methods to predict such demands for the next day. While the 1-day-back forecast method led to the prediction of heat demand, the N-day-average performed best for DHW demand when Unbiased Exponentially Moving Average (UEMA) is used with a memory of 2.5 days. This is surprising as these forecasting methods are very simple and do not leverage additional information sources such as weather forecasts.

1 Introduction

Energy optimization for single family homes requires predictions of future energy demands, typically for heat, domestic hot water (DHW), and electrical power. An example is the control of a heat pump which is fueled by the power grid and photovoltaic energy from the roof top. The latter should be well utilized, but switching cycles for the heat pump must be kept low to ensure a long lifetime of the heat pump. This optimization problem requires knowledge of the energy demand at least one day in advance.

While it is easy to predict energy demands for large neighborhoods consisting of hundreds of units, it is harder for single family homes. In this work, we evaluate the appropriateness of different forecasting methods for this task. We utilize them to create models from historical energy data of a single family home and predict those demands based on these models. We compare the suitability of the forecasting approaches by the error between their predictions and the historical data.

The paper is structured as follows. Section 2 gives an overview of related work. In Section 3 we describe the studied forecasting methods. Section 4 presents and investigates the data set. An evaluation of the forecasting methods is presented in Section 5. We conclude this work in Section 6.

2 Related Work

Aydinalp et al. developed and evaluated a neural network approach to model residential energy consumption in their paper [5]. They adopted their model to heat and DHW demand and evaluated the accuracy of the predictions. They focused on the construction of a good neural network model and on features, but did not compare different approaches.

Lomet et al. [3] investigated the DHW consumption of single family homes. They analysed real data from such housing units and developed an ARMA model to forecast DHW demands. Their results indicate that this type of model could be suitable to forecast such demands, but they have not performed evaluations to compare different forecasting approaches for DHW demands.

Idowu et al. [4] analysed DHW and heat demand for multi-family apartments. Based on their analysis, forecasts were computed using supervised machine learning approaches. They concluded that using super-vector regression leads to least errors, but their evaluation lacks comparison with simpler approaches like 1-day-back.

Idowu et al. [6] also evaluated and compared more recently multiple advanced machine learning approaches to forecast heat and DHW demand in residential and industrial buildings. Their evaluations show that support vector machines are well suited to forecast both DHW and heat. However, also these evaluations lack inclusion of simpler approaches like 1-day-back.

3 Forecasting Methods

In this section, we give a short primer on the studied forecasting methods. Forecasting predicts values for a primary time series. Historical data for that time series may be used to estimate most probable future values. Formally, forecasting the t-th value $\hat{y}_t$ of a known discrete time series $y_0, y_1, .., y_{t-1}$ is the computation of some forecasting function

$$\hat{y}_t = F(y_0, y_1, .., y_{t-1}).$$

Sometimes forecasting is not only based on historical data of the primary time series but also on a secondary time series $x_0, x_1, ..., x_t$ which is correlated with the primary time series. Thereby, $x_0, x_1, ..., x_{t-1}$ is historical and x_t is predicted. For example, the demand for heat (primary time series) may strongly correlate with the outside temperature

(secondary time series). Then, the secondary time series helps to forecast the desired value $\hat{y}_t$ of the primary time series:

$$\hat{y}_t = F(y_0, y_1, .., y_{t-1}, x_0, ..., x_t).$$

Different forecasting methods can be used to compute $\hat{y}_t$. In the following, we describe several simple forecasting methods that we evaluate later in this paper.

3.1 N-Day-Back

The N-day-back forecast method utilizes the time time series of the N^{th} preceding day (historical data) as forecast of the next day. Let n be the number of data points per day. Then the forecasting function can be described as $F(y_0, y_1, ..., y_{t-1}) := y_{t-(N \cdot n)}$. A special case is the 1-day-back forecast method which uses the historical time series of the previous day as forecast of the next day.

3.2 N-Day-Average

For the N-day-average method, we take the average demand of N preceding days as a demand forecast of the next day. Thereby, we utilize two different averaging methods presented in [1].

3.2.1 Window Moving Average (WMA)

WMA defines a window containing the last W data points and computes their arithmetic mean. Thus, W is an integral value. In this work, we apply WMA to daily demands and compute the average demand of the last $W = N$ days. A memory $M = W \cdot \Delta t$ can be defined where Δt is the inter-sample distance, i.e., a day in our context. The memory expresses the time over which a sample is remembered in the moving average.

3.2.2 Unbiased Exponential Moving Average (UEMA)

UEMA can be calculated by $A_t = \frac{S_t}{N_t}$ where the weighted sum S_t and the weighted number N_t are recursively defined as

$$S_t = \begin{cases} X_0 & t = 0 \\ a \cdot S_{t-i} + X_t & \text{otherwise} \end{cases}$$

$$N_t = \begin{cases} 1 & t = 0 \\ a \cdot N_{t-i} + 1 & \text{otherwise.} \end{cases}$$

Past values of the original time series X_t contribute to all future average values A_t. However, their impact decreases exponentially over time. The parameter a determines the memory of the moving average by $M = \frac{\Delta t}{1-a}$. Again, the memory cannot be smaller than the inter-sample distance Δt. However, any larger memory M is possible with $a = 1 - \frac{\Delta t}{M}$. In the context of N-day-average we set the memory to $M = N$. Therefore, N does not need to be an integral value if UEMA is used for averaging.

3.3 Linear Regression (LR)

LR [7] describes the dependency of the primary time series $y_0, y_1, ..., y_{t-1}$ on the secondary time series $x_0, x_1, ..., x_t$ with a linear function $f(x) := \beta_0 + \beta_1 \cdot x$. This may be useful if both time series are highly correlated. The forecast value $\hat{y}_t$ is then the mapping of x_t under this linear function. Mostly LR finds a best-fit line that minimizes the sum of squared distances for a set of data points $(x_i, y_i)_{0 \le i < n}$. This line is efficient to compute by a compact formula. However, our comparative metric in Section 5 is the absolute average error $\frac{1}{n} \sum_{0 \le i < n} |f(x_i) - y_i|$ between the best-fit line f and a set of data points. Therefore, we prefer a simple numerical method based on nested intervals to derive appropriate parameters β_0 and β_1 that minimizes $\frac{1}{n} \sum_{0 \le i < n} |f(x_i) - y_i|$.

3.4 Bounded LR (BLR)

As the LR-based best-fit line yields negative heat demands for high outside temperatures in Section 4.1, we propose BLR. It utilizes a best-fit line, but yields zero instead of negative values. Also for BLR we compute the parameters β_0 and β_1 for a best-fit line by minimizing the sum of absolute errors.

3.5 Daytime-Specific BLR

We will first apply LR and BLR based on entire days, i.e., we compute a best-fit line that takes the average daily temperature as input and yields the daily energy demand as output. As an alternative, we will apply LR and BLR based on the average temperature of the daily time intervals 0-6, 6-12, 12-18, and 18-24 o'clock and predict their energy demands. Finally, we sum up the predicted energy demands over a day to obtain the daily energy demand.

3.6 Smoothing

The dependency of the forecast time series on the prediction time series may be time-delayed. For example, the temperature in a building does not fall immediately when it becomes cool outside, in particular in case of good insulation. As a consequence, average temperatures smoothed over time may yield better forecasts for energy demands. Therefore, we propose to apply the LR and BLR methods based on smoothed historical data. We use UEMA (see Section 3.2.2) to smooth the data series, i.e., we use the average value A_t instead of X_t.

Smoothing variant	Time series	
	Primary	Secondary
no-smoothing	original	original
x-smoothing	smoothed	original
y-smoothing	original	smoothed
xy-smoothing	smoothed	smoothed

Table 1 The smoothing variant determines whether model parameters are calculated based on original or smoothed times series.

We now explain several smoothing variants for LR/BLR forecasting. Secondary and/or primary historical time series may be smoothed for forecasting. The linear functions for the LR/BLR model are computed based on either original or smoothed time series. Table 1 shows 4 variants and

the time series based on which the parameters for the best-fit line are fitted.

For prediction purposes, the original secondary time series are utilized for no-smoothing and x-smoothing. For y-smoothing and xy-smoothing, the secondary time series $x_0, x_1, ..., x_t$ is smoothed including the predicted value x_t for which the corresponding value $\hat{y}_t$ of the primary time series is to be forecast based on the computed LR/BLR model.

4 Data Analysis

The data set used for the evaluation in Section 5 contains real data from a single family home near Düsseldorf of the year 2018. The data set consists of time series for PV production, outside temperature, heat demand, DHW demand, and other electrical demand. The resolution is one data point per minute. In contrast, our objective in Section 5 is to forecast the heat and DHW demand for the entire next day.

Figure 1 illustrates the data while temperature is averaged and demands are accumulated per day. The temperature is low in winter and high in summer with considerable oscillations throughout the year. In the following, we analyse the data for heat and DHW demand in more detail.

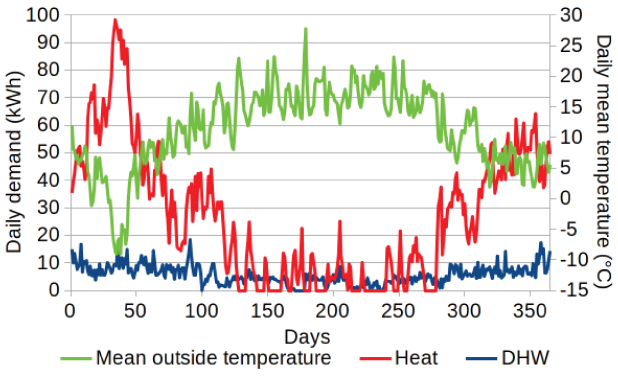
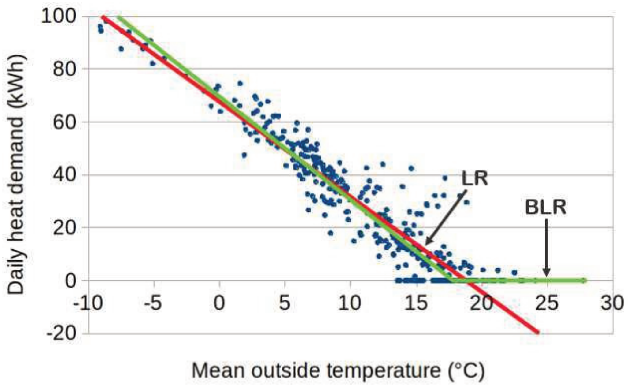


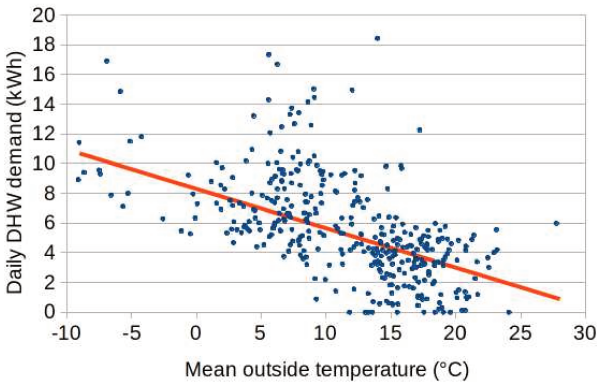
Figure 1 Daily heat and DHW demand together with mean outside temperature per day in 2018.

4.1 Heat

Figure 1 shows that the energy demand for heat is a multiple of the one for DHW. In addition, we clearly see that it strongly depends on the outside temperature. We demonstrate this phenomenon by the scatter plot in Figure 2a. The coordinates of the points consist of the average temperature of a single day and the corresponding heat demand. The points are clustered along a line and we recognize the obvious trend that heat demand decreases with increasing outside temperature. This proposes a linear relation between daily heat demand and mean outside temperature. We obtain the the best-fit lines for LR with the parameters $\beta_0 = 66.830$ and $\beta_1 = -3.484$ and for BLR with the parameters $\beta_0 = 69.600$ and $\beta_1 = -3.910$. These lines minimize the average deviation to the data points. They are also illustrated in Figure 2a. In contrast to LR, the BLR best-fit line does not yield negative heat demands.



(a) Heat demand with best-fit lines for LR and BLR.



(b) DHW demand with best-fit line for LR.

Figure 2 Impact of mean outside temperature on the daily heat and DHW demand.

4.2 Domestic Hot Water (DHW)

Figure 1 shows that the demand for DHW is rather low over the year compared to the demand for heat. It oscillates over days and there is some seasonal impact.

We provide a scatter plot in Figure 2b. The points are clustered, but they are more distributed than in 2a, i.e., the linear dependency of the DHW demand on the outside temperature is weaker than the one of heat. The figure also shows the LR best-fit line which is obtained for $\beta_0 = 9.122$ and $\beta_1 = -0.305$. As this line does not yield negative values in the relevant range, there is no need for BLR.

5 Evaluation

In this section we evaluate several forecasting methods for heat and DHW demand. First, we describe the evaluation methodology. Then, we apply the forecasting methods presented in Section 3 to the data set presented in Section 4 in order to compare their predicted heat and DHW demands.

5.1 Methodology

We predict the per-day heat and DHW demand for all days of 2018 in the data set. To assess the accuracy of the forecasting methods, we compute the difference of the real demands in the data set and the forecast values, and provide

absolute and relative average errors.
The N-day-back and the N-day-average methods can be applied to the data set just after aggregating the demands for entire days.
Parameter fitting is not needed for N-day-back and for N-day-average. This is different for LR and BLR. For these methods we derived the best-fit lines based on the entire data set as presented in Section 4. We use it to predict the demand for a specific day based on its average outside temperatures. That is, we predict the data based on which we calibrated the LR and BLR model. We do that due to lack of sufficient data. This is not feasible in practice and rather overrates the quality of the forecast. Nevertheless, we will see that these methods are outperformed by simpler methods.
For N-day-back, N-day-average, or smoothing variants, preceding days may be missing at the beginning of the year. Then we take the days at the end of the year as a substitute in a cyclic manner.

5.2 Heat

We consider forecast for heat demands. Table 2 compiles the absolute and relative average forecast errors to quantify the accuracy of various forecasting methods.

Forecast method	Abs. avg. error (kWh)	Rel. avg. error (%)
1-day-back	4.303	15.9
3-day-back	7.723	28.5
7-day-back	9.692	35.7
Linear regression	6.226	22.9
Bounded LR	5.232	19.3
Daytime-sp. BLR	5.375	19.8

Table 2 Absolute and relative average errors for forecasts of daily heat demands; the average demand is 27.15 kWh.

The 1-day-back method leads to the least forecast errors, followed by BLR and the daytime-specific BLR method. The other methods cause significantly larger forecast errors. The 7-day-back method performs particularly badly. We tested that method as we suspected that weekly patterns in human behaviour could have a measurable impact on heat demand.
BLR clearly outperforms linear regression because it does not forecast negative values for high outside temperatures.
For daytime-specific BLR we obtained daytime-specific best-fit lines with the parameters given in 3. However, daytime-specific BLR is worse than normal BLR so that its complexity does not pay off.

Interval	β_0	β_1
0-6	14.2	-1.203
6-12	25.5	-1.269
12-18	27.8	-1.536
18-0	6.2	-0.408

Table 3 Parameters for best-fit lines for BLR-based daytime-specific forecasts.

We consider N-day-average whose forecast errors are compiled in Table 4, both for WMA and for UEMA as averaging methods. N-day-average with a memory of a single day yields 1-day-back, therefore, we see the same errors. Values for WMA can be computed only for memories that are multiples of entire days. Increasing the memory degrades predictions of heat demands both for WMA and UEMA so that N-day-average is not useful compared to 1-day-back.

Memory (d)	WMA		UEMA	
	Abs. avg. error (kWh)	Rel. avg. error (%)	Abs. avg. error (kWh)	Rel. avg. error (%)
1	4.303	15.9	4.303	15.9
1.5	-	-	4.682	17.2
2	4.990	18.4	5.053	18.6
2.5	-	-	5.362	19.8
3	5.579	20.6	5.618	20.7

Table 4 Absolute and relative average errors for forecasts of daily heat demand using the N-day-average method with WMA and UEMA.

We investigate the potential of the smoothing variants mentioned in Table 1 to improve forecasts. For no-smoothing and y-smoothing, we take the outside temperatures of the same day in the data set as x-input for the BLR model. In case of x-smoothing and xy-smoothing, we compute a time series for smoothed temperatures in 2018 and use the smoothed temperature of the corresponding day as x-input for the BLR model.
Table 5 provides forecast results for different memories. A memory of 1 day means no smoothing. We observe that no smoothing variant improves the forecast of the simple BLR method. In contrast, increasing memory degrades forecasting results.

Memory (d)	x-smoothing (%)	y-smoothing (%)	xy-smoothing (%)
1	19.3	19.3	19.3
2	19.4	19.4	19.4
3	19.8	19.8	19.5
4	20.1	20.0	19.5

Table 5 Relative error for the forecasts of daily heat demand; different memories are considered for x, y, and xy-smoothing in combination with BLR.

5.3 Domestic Hot Water (DHW)

We consider forecast of DHW demands. Table 6 compiles the absolute and relative average errors for various forecasting methods. Again, the 1-day-back method is best, followed by LR. The 3- and 7-day-back methods do not perform well. As the best-fit line for LR does not yield negative values, there is no need for BLR as LR and BLR lead to the same best-fit line under such conditions.

Forecast method	Abs. avg. error (kWh)	Rel. avg. error (%)
1-day-back	1.867	33.0
3-day-back	2.222	39.3
7-day-back	2.437	43.1
Linear regression	2.091	37.0

Table 6 Absolute and relative average error for forecasts of daily DHW demand; the average demand is 5.653 kWh.

We have also experimented with all the presented smoothing variants but without any improvement up to large memories of 84 days.

Finally, we consider N-day-average as forecasting method. Table 7 presents the absolute and relative average errors for this method. The forecast errors depend on the specific averaging method, i.e., WMA or UEMA, and the chosen memory. UEMA yields better forecasts than WMA and the best memories are 2.5 days for UEMA and 3 days for WMA. Both methods clearly outperform even 1-day-back so that UEMA leads to the best forecasting results for DHW.

Memory (d)	WMA		UEMA	
	Abs. avg. error (kWh)	Rel. avg. error (%)	Abs. avg. error (kWh)	Rel. avg. error (%)
1	1.867	33.0	1.867	33.0
1.5	-	-	1.702	30.1
2	1.740	30.8	1.672	29.6
2.5	-	-	1.666	29.5
3	1.724	30.5	1.678	29.7
3.5	-	-	1.692	29.9
4	1.751	31.0	1.708	30.2
4.5	-	-	1.721	30.4
5	1.757	31.1	1.733	30.6
6	1.777	31.4	1.756	31.1
7	1.793	31.7	1.774	31.4
14	1.936	34.3	1.871	33.1
21	1.936	34.3	1.935	34.2
28	1.967	34.8	2.002	35.4

Table 7 Absolute and relative average error for the forecasts of daily DHW demand using the N-day-average method with WMA and UEMA.

6 Conclusion

We compared different methods to predict the demand for heat and domestic hot water (DHW) for the next day in a single family home.

To predict heat demand, we obtained the best results with the very simple 1-day-back method. It clearly outperformed linear regression (LR), bounded LR (BLR), and daytime-specific BLR. This is surprising as 1-day-back does not take advantage of available weather forecast, which is in contrast to some other methods. Averaging methods led to worse results. Smoothing-based BLR could not achieve any improvement with regard to BLR.

To forecast DHW demand, 1-day-back again outperformed LR including smoothing methods. However, the N-day-average led to even better results for small memories. We obtained the best forecasts for the UEMA averaging method with a memory of 2.5 days. This is again surprising as N-day-average does not leverage additional information like weather forecast, either.

Although we have identified best forecast methods for heat and DHW including parameters, we point out that these results have been gained from a single single family home in 2018. It would be helpful to validate our findings on a larger data set, over a longer duration, and for houses with different energy demands. Moreover, it would be interesting to consider aggregated demands from multiple houses or blocks of flats. The use of machine learning approaches is certainly also worthwhile to study provided sufficient data is available.

7 Literatur

[1] M. Menth and F. Hauser, “On Moving Averages, Histograms and Time-Dependentrates for Online Measurement”, in Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, 2017, pp. 103–114.

[2] J. Ferrero Bermejo, J. F. Gomez Fernandez, F. Olivencia Polo, and A. Crespo Márquez, “A Review of the Use of Artificial Neural Network Models for Energy and Reliability Prediction. A Study of the Solar Pv, Hydraulic and Wind Energy Sources”, Applied Sciences, vol. 9, no. 9, pp. 1844-1862, 2019.

[3] A. Lomet, F. Suard, and D. Chèze, “Statistical Modeling for Real Domestic Hot Water Consumption Forecasting", Energy Procedia, vol. 70, pp. 379–387, 2015.

[4] S. Idowu, S. Saguna, C. Åhlund and O. Schelén, "Forecasting Heat Load for Smart District Heating Systems: A Machine Learning Approach", 2014 IEEE International Conference on Smart Grid Communications (SmartGridComm), Venice, 2014, pp. 554-559.

[5] M. Aydinalp, V. Ismet Ugursal, and A. S. Fung, “Modeling of the Space and Domestic Hot-Water Heating Energy-Consumption in the Residential Sector Using Neural Networks”, Applied Energy, vol. 79, no. 2, pp. 159–178, 2004.

[6] S. Idowu, S. Saguna, C. Åhlund, and O. Schelén, “Applied Aachine Learning: Forecasting Heat Load in District Heating System”, Energy and Buildings, vol. 133, pp. 478–488, 2016.

[7] R. J. Hyndman and G. Athanasopoulos, “Forecasting: Principles and Practice”, OTexts, 2nd ed., 2018.

3.2 Load Profile Negotiation for Compliance with Power Limits in Day-Ahead Planning

Load Profile Negotiation for Compliance with Power Limits in Day-Ahead Planning

Florian Heimgaertner, Sascha Heider, Thomas Stueber, Daniel Merling, and Michael Menth
Chair of Communication Networks, University of Tuebingen, Tuebingen, Germany
Email: {florian.heimgaertner,thomas.stueber,daniel.merling,menth}@uni-tuebingen.de, sascha.heider@student.uni-tuebingen.de

***Abstract*—The variability of electrical energy prices at the spot market incentivizes cost-optimized load scheduling. Based on day-ahead price forecasts, energy costs can be considerably reduced by shifting energy-intensive processes to times with lower energy prices. While the mechanism of the market match demand and supply, they currently do not consider technical limitations of the electrical power grid. A large number of consumers scheduling electrical loads according to the same price forecast could result in congestion in the transmission or distribution systems.**

We propose a mechanism for day-ahead scheduling that enables negotiation of load profiles between multiple consumers and an aggregator in compliance with overall power limits. We present two mechanisms for an aggregator without knowledge about internal details of the participants to achieve this goal and compare the performance to the results of a centralized scheduler with global knowledge.

***Index Terms*—Smart grids, demand-side management, scheduling, virtual power plant.**

I. Introduction

The increasing share of weather-dependent renewable power generation leads to a large intraday variability of wholesale energy prices. Shifting loads to times with lower energy prices can considerably reduce energy costs and helps to increase the use of renewable energy by improving the match of demand and supply. Schedules of multiple consumers optimized for the same price forecast can lead to extreme load peaks. The mechanisms of the energy markets match the demand peaks and the production peaks, so the optimization of schedules based on price forecasts could be beneficial for both the generation and consumption domains. However, it can lead to problems in the transmission and distribution domains as there is no guarantee that the physical grid is capable of transporting the purchased energy volumes from generators to the consumers.

We proposed a distributed control architecture for virtual power plants [1] where participating enterprises locally optimize their load schedules according to price forecasts provided by an aggregator. The aggregator trades energy at the spot market on behalf of the participating enterprises. However, if the combined load profiles of a set of enterprises violate any constraints, the aggregator needs to negotiate re-scheduling with the affected enterprises.

In this work we propose mechanisms for a set of business units to negotiate load profiles that reduce energy costs while avoiding the violation of restrictions imposed by bottlenecks in the power grid.

This paper is structured as follows. Section II discusses related work. In Section III we present the context for the optimization and an abstract model for enterprises with load shifting capabilities. Section IV proposes two mechanisms for load profile negotiation. In Section V we shows the scenario and parameters for the evaluation and in Section VI we evaluate the performance of the negotiation mechanism and compare its results to an centralized scheduling approach with global knowledge. Section VII concludes the paper.

II. Related Work

Ibars et. al. present a distributed load management using dynamic pricing [2]. The approach is based on a network congestion game. The authors show that the system converges to a stable equilibrium. Biegel et. al. [3] describe a receding horizon control approach for moving shifting loads to minimize costs for balancing energy while avoiding grid congestion. Huang et. al. [4] propose a congestion management method for distribution grids with a high penetration of electrical vehicles and heat pumps. They use a decomposition-based optimization. In [5] they present a real-time approach for congestion management using flexible demand swap. Boroojeni et. al. [6] propose an oblivious routing economic dispatch approach for distribution grids. Bagemihl et. al. [7] describe a market-based approach to increase the capacity of a distribution grid without physical grid expansion. Hazra et. al. [8] propose a demand-response mechanism for grid congestion management using ant colony optimization. Sundström and Binding [9] propose a method for the optimization of charging schedules for electric vehicles while avoiding grid congestion.

Most work in the area of grid congestion management is based on actual grid topologies and focuses on global optimizations to avoid grid congestion. This paper uses a simplified approach, limiting congestion to a single bottleneck and focuses on interactive negotiation without global knowledge.

III. Model

In this section, we present the use case. We explain the concept of load profiles and define the parameters for the consumer model.

A. Use Case

The grid connection of a consumer is limited in electrical power by technical or contractual means. We denote this limit as l_c where c is a consumer. Due to limitations in the distribution grid, similar restrictions apply to groups of consumers, e.g., urban districts. As the sum of all individual power limits can be larger than the limit for the group, a group of consumers could exceed the group power limit L while still complying with their individual limits, i.e., $\sum_{c \in \mathcal{C}} l_c > L$ where $\mathcal{C}$ is a set of consumers. This problem becomes more severe in presence of price-optimized day-ahead planning when loads of all flexible consumers are scheduled for the times with the lowest energy price forecasts. However, day-ahead planning usually involves an aggregator providing the forecasts and trading at the energy markets. As an aggregator requires load forecasts of all aggregated consumers, we propose a mechanism for day-ahead demand-side management (DSM) within the group the aggregated consumers.

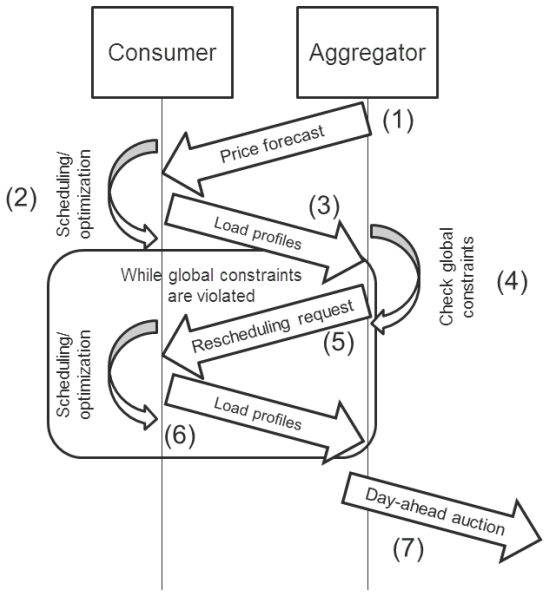


Fig. 1: Negotiation process between enterprise and aggregator during day-ahead planning.

Figure 1 shows the negotiation process to ensure that limitations for a group of consumers are complied with. The aggregator distributes price forecasts for the day-ahead energy market to the aggregated consumers (1). Each consumer computes price-optimized schedules based on their model parameters using the price forecast received from the aggregator (2). After the best schedule is selected, the consumers send the load profiles to the aggregator (3). After receiving load profiles from the consumers, the aggregator checks global constraints (4). An example for a global constraint is a cumulative power limit for a group of participants imposed by the grid operator. If such a constraint is violated, the aggregator sends a rescheduling request to the affected groups or individual participants (5). The affected consumers perform planning and optimization based on additional information provided by the aggregator and submit new load profiles (6). Steps (4)–(6) are repeated until the global constraints are no longer violated. Finally, energy is traded at the day-ahead market (7).

B. Consumer Model and Load Profiles

A load profile is a time series of electrical load over a given period. As we focus on day-ahead optimization, we chose a period of 24 hours and a granularity of one hour. A time slot is denoted as t and the set of the time slots of a day is defined as $\mathcal{T} := \{0, \ldots, 23\}$. We denote the load profile of a consumer c as e_c^t, $t \in \mathcal{T}$, with an energy demand for each hour of a day. The total energy demand of all consumers in time slot t is limited by the group power limit L^t.

For our study we use an abstract model of a business consumer with flexibility for load shifting. We do not consider internal organization and dependencies among processes within a consumer, but limit the model to energy and cost parameters. The consumer is defined by a daily demand of electrical energy E_c, a power limit l_c^t, and operational costs A_c^t. The objective is to find a set of load profiles $e_c^t, t \in \mathcal{T}$, that satisfy the following conditions.

$$\sum_{t \in \mathcal{T}} e_c^t = E_c \; \forall c \in \mathcal{C} \tag{1}$$

$$e_c^t \leq l_c^t \; \forall t \in \mathcal{T}, c \in \mathcal{C} \tag{2}$$

$$\sum_{c \in \mathcal{C}} e_c^t \leq L^t \; \forall t \in \mathcal{T} \tag{3}$$

Each load profile is associated with costs. F^t is the energy price forecast for time slot t. A_c^t gives the additional (non-energy) operation costs of a consumer c in time slot t. The total costs C_c for a consumer c are given by

$$C_c = \sum_{t \in \mathcal{T}} e_c^t \cdot F^t + A_c^t. \tag{4}$$

IV. Mechanisms

In this section, we present a linear program that computes load profiles for each participant resulting in the lowest total costs while complying with the group power limit. The linear program needs global knowledge, i.e., it requires information about internal details such as cost structures of all participants to compute the solution. However, aggregator operation without such global knowledge of internal details about the participating enterprises is an explicit goal of [1]. Therefore, we propose two methods for load profile negotiation that work without global knowledge. The sequential approval method is based on a first-come-first-serve approach combined with a compensation for swapping time slots. The simultaneous approval method requests multiple load profiles per participant to find an acceptable combination of load profiles.

A. Load Optimization Using Global Knowledge

The load profiles $e_c^t, t \in \mathcal{T}, c \in \mathcal{C}$ consist of continuous variables that can be determined by the following linear program.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{t=0}^{23} \sum_{c \in \mathcal{C}} F^t e_c^t + A_c^t \\
\text{subject to} \quad & \sum_{c \in \mathcal{C}} e_c^t \leq L^t, && t \in \mathcal{T} \\
& \sum_{t=0}^{23} e_c^t = E_c, && c \in \mathcal{C} \\
& e_c^t \leq l_c^t, && t \in \mathcal{T}, c \in \mathcal{C} \\
& e_c^t \in \mathbb{R}, && t \in \mathcal{T}, c \in \mathcal{C}
\end{aligned}
$$

B. Sequential Approval of Load Profiles

For the sequential approval method, each submitted load profile is individually approved after submission unless its load combined with the previously approved load profiles would exceed the group power limit. To resolve the violation, all participants with acknowledged energy demand in the respective time intervals compute alternative load profiles avoiding the overloaded time slots $t \in \mathcal{T}'$. They submit load profiles annotated with the additional costs resulting from higher energy prices or increased operation costs in alternative time intervals. The aggregator selects the combination of load profiles with the lowest total additional costs. The process is repeated until a load profile for each participant is approved.

A linear program is used to find an appropriate combination of load profiles. The load profiles are selected such that the sum of the additional costs, i.e., the differences between the respective cheapest load profiles, of all enterprises is minimized. If every consumer c hands in n_c load profiles, let x_c^i be a binary variable which is true iff the i-th schedule of enterprise $c \in \mathcal{C}$ is selected. Furthermore, let $e_c^{t,i}$ be the energy demand of load profile i of consumer c in slot t, C_c^i the total cost of consumer c for load profile i and L^t the group power limit of slot t.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{c \in \mathcal{C}} \sum_{i=1}^{n_c} (C_c^i - C_c^1) \cdot x_c^i \\
\text{subject to} \quad & \sum_{i=1}^{n_c} x_c^i = 1, && c \in \mathcal{C} \\
& \sum_{c \in \mathcal{C}} \sum_{i=1}^{n_c} e_c^{t,i} x_c^i \leq L^t, && t \in \mathcal{T} \\
& x_c^i \in \{0,1\}, && c \in \mathcal{C}, i = 1, ..., n_c
\end{aligned}
$$

The inequations ensure that every consumer has exactly one schedule approved and that the group power limit is not exceeded in any time slot.

The participant triggering the violation compensates additional costs for participants with approved load profiles or selects a different load profile if costs are lower compared to the required compensation. While a participant can exaggerate the additional costs to generate additional revenue from rescheduling, higher costs lead to a lower chance for a load profile to be selected by the aggregator or accepted by the participant that triggers the violation.

C. Simultaneous Approval of Load Profiles

For the sequential approach the order of load profile submissions is important. Therefore late submissions of load profiles are penalized and the cost increase is distributed unevenly among the participants. This might lead to acceptance problems and prevent some enterprises from participating.

A straightforward implementation of an order-agnostic negotiation method consists of iterative energy price increases for the overloaded time slots and requests for new load profiles from all participants. However, this approach leads to artificially high energy prices and experiments showed that it fails to resolve violations for low group power limits while the sequential approval method still succeeds. Therefore, we propose a simultaneous approval method that works without modified price forecasts.

The aggregator checks for limit violations after all participants have submitted load profiles. In case of a limit violation the aggregator requests an alternative schedule from all participants, indicating the affected time slots $t \in \mathcal{T}'$. With the original load profiles and the alternative load profiles, the aggregator computes a combination not exceeding the limits. If such a combination does not exist, the aggregator repeatedly increases the number of requested load profiles per participant until there is a combination of load profiles that complies with the limits. The participants annotate the list of submitted load profiles with a preference.

The optimal selection of load profiles is computed using a linear program. If every consumer hands in n load profiles, let x_c^i be a binary variable which is true iff the i-th schedule of enterprise $c \in \mathcal{C}$ is selected. Furthermore, let $e_c^{t,i}$ be the energy demand of load profile i of consumer c in time slot t, C_c^i the total cost of consumer c for load profile i and L^t the group power limit of slot t.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{c \in \mathcal{C}} \sum_{i=1}^{n} i \cdot x_c^i \\
\text{subject to} \quad & \sum_{i=1}^{n} x_c^i = 1, && c \in \mathcal{C} \\
& \sum_{c \in \mathcal{C}} \sum_{i=1}^{n} e_c^{t,i} x_c^i \leq L^t, && t \in \mathcal{T} \\
& x_c^i \in \{0,1\}, && c \in \mathcal{C}, i = 1, ..., n
\end{aligned}
$$

The weighting of load profiles by the number i gives the load profiles a preference by the order of submission. The consumer $c \in \mathcal{C}$ indicates that a load profile $e_c^{t,i}$ is preferred over a load profile $e_c^{t,i+1}$.

V. Evaluation Model

In this section we describe company-specific operational costs and day-ahead forecasts used in our experiments. Finally, we point out how load profiles are calculated for companies that participate in the negotiation processes described in Section IV-B and Section IV-C.

A. Operational Cost Factor

In the model described in Section III-B operating costs A_c^t can be given per time slot for each consumer. For our evaluation, we model the A_c^t as a dependency of the energy demand e_c^t and an operating cost factor $f_{o,c}^t$. We model the operational cost factor $f^t o, c$ of a consumer c using an interval of primary business hours and two intervals of secondary business hours. The primary business hours start at time slot t_c^p and its duration is d_c^p time slots. The secondary business hours are d_c^s time slots before and after the the primary business hours. During the primary business hours the operational cost factor is $f_{o,c}$ and $2 \cdot f_{o,c}$ during the secondary business hours. Outside of primary and secondary business hours operational costs are infinite, so business operation is not possible. The additional operational costs are given by the operational cost factor and the energy demand in the respective time slot: $A_c^t = f_{o,c}^t \cdot e_c^t$.

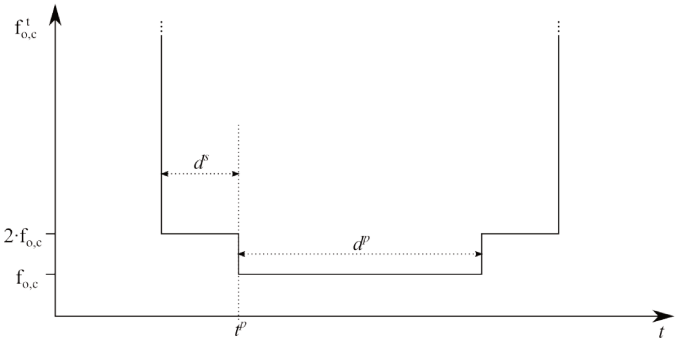


Fig. 2: Operation cost factor $f_{o,c}^t$ defined by parameters t_c^p, d_c^p, d_c^s, and n.

An example of operating costs over time defined by those parameters is given in Figure 2. The operating costs are twice as large during secondary business hours compared to primary business hours. During nonproductive hours, operating costs are infinite.

For our evaluation we chose $t_c^p \in \{7, \ldots, 11\}$, $d_c^p = 8$, and $d_s^c = 2$. We define four classes of consumers by $(E_c, f_{o,c})$, $E_c \in \{1200\,\text{kWh}, 3000\,\text{kWh}\}$ and $f_{o,c} \in \{500\,€/\text{MWh}, 1000\,€/\text{MWh}\}$. The individual power limit l_c^t is set to $\frac{E_c}{6}$ in all time slots. Each starting time slot t_c^p is used once per class resulting in a group size of 20.

B. Day-Ahead Price Forecast

The prices shown in Figure 3 are used as day-ahead price forecast. While the actual prices are fictitious, the price level and the development over the 24 hour period are typical for the German day-ahead energy market.

C. Local Load Scheduling

The total costs of a schedule arise from the energy costs associated with the load profile and the operation costs. The price forecast is given as F^t, $t \in \mathcal{T}$, where F^t is the predicted price per MWh during time slot t.

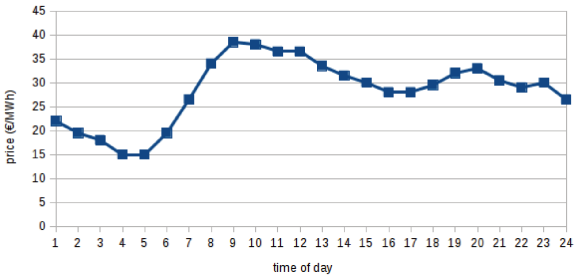


Fig. 3: Day-ahead energy price forecast.

Data: T_c^t for $0 \leq t < 24$, l_c^t, E_c
Result: e_c^t for $0 \leq t < 24$
$t[] :=$ list of times sorted ascending by value of T_c^t
$i = 0$
$E := E_c$
while $E > 0$ **do**
  if $E > l_c^t$ **then**
    $e_c^{t[i]} := l_c^t$
    $E := E - l_c^t$
  else
    $e_c^{t[i]} := E$
    $E := 0$
  $i := i + 1$
end

Algorithm 1: Cost-optimized local load scheduling.

As the hourly operation costs A_c^t in our scenario depend on the energy consumption the algorithm for producing cost-optimized schedules is straightforward. The scheduling is implemented using a greedy approach as shown in Algorithm 1. A scheduler first computes the total operation costs per kWh $T_c^t = F^t + f_{o,c}^t c$. At the time t with the lowest T_c^t, energy consumption e_c^t is set to the maximum allowed by l_c^t, proceeding with the second-lowest T_c^t and so on until $\sum_{t=0}^{23} e_c^t = E_c$. The total cost C_c of a schedule i is computed according to Equation (4).

For the computation of alternative load profiles, the consumers repeat Algorithm 1 with selectively reduced l_c^t for the affected time slots $t \in \mathcal{T}'$. For the sequential approval method, the consumers use $l_c^t = 0, \forall t \in \mathcal{T}'$. For the simultaneous approval method, the consumers reduce l_c^t for the affected time slots $t \in \mathcal{T}'$ by 1% iteratively.

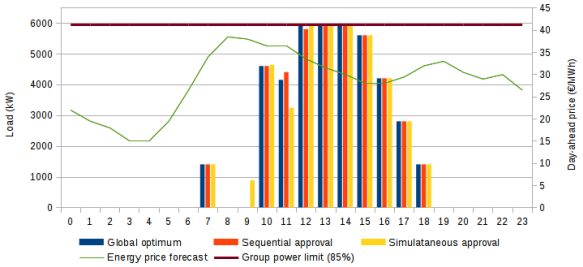
VI. Results

In this section we present the results of the evaluation. We show the load profiles resulting from sequential and simultaneous approval and compare them to the global optimum. In the evaluation scenario described in Section V, the sum of all individual power limits is given by $\sum_{c \in \mathcal{C}} l_c^t = 7000\,\text{kW}\ \forall t \in \mathcal{T}$. We use relative group power limits of 85%, 65%, and 55%, corresponding to $L^t \in \{5950\,\text{kW}, 4550\,\text{kW}, 3850\,\text{kW}\}$ for all time slots. We show the cost increase compared to each

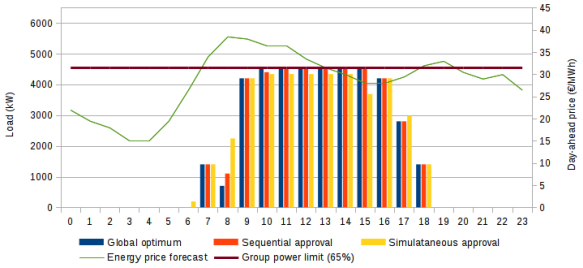
TABLE I: Relative total cost increase.

	Group power limit		
	5950 kW (85%)	4550 kW (65%)	3850 kW (55%)
Global optimum	0.03%	0.15%	4.40%
Sequential approval	0.07%	0.23%	6.01%
Simultaneous approval	0.04%	1.02%	12.12%

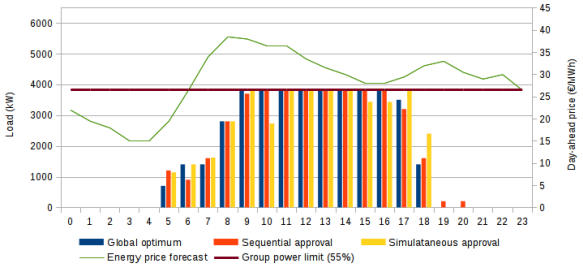
consumer's preferred load profile, which would be possible with a group power limit of $L^t = 7000\,\text{kW}$. Finally we give an overview on the scheduling overhead caused by both mechanisms.



(a) Results for 85% relative group power limit.



(b) Results for 65% relative group power limit.

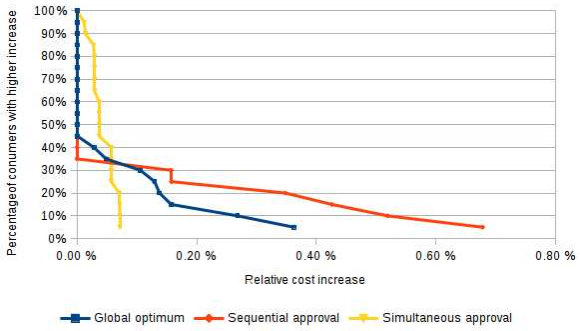


(c) Results for 55% relative group power limit.

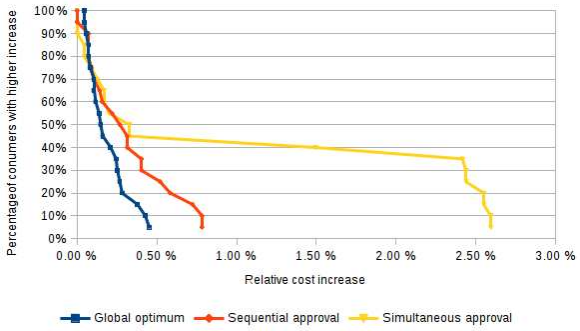
Fig. 4: Load profiles resulting from simultaneous approval, sequential approval, and global optimization at different group power limits.

A. Negotiation Results at 85% Relative Group Power Limit

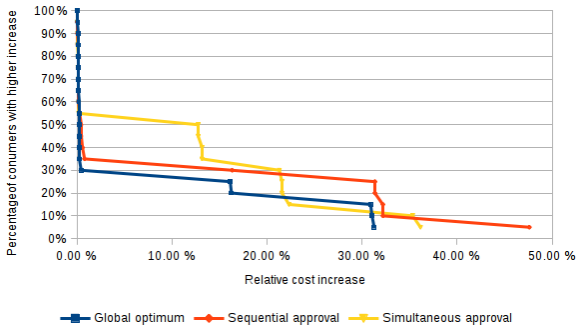
The results for the load profile negotiation at a group power limit of 5950 kW are shown in Figure 4(a). Both the sequential and simultaneous approval methods yield load profiles similar to the global optimum. The only major difference can be seen at the 9:00 time slot which is only selected in the simultaneous approval method. However, Table I shows only minimal differences regarding the increased costs. While the difference is negligible, the simultaneous approval method actually leads



(a) Results for 85% relative group power limit.



(b) Results for 65% relative group power limit.



(c) Results for 55% relative group power limit.

Fig. 5: Percentage of consumers with higher relative cost increase at different group power limits.

to lower increased costs compared to the sequential approval method. Figure 5(a) shows that no cost increase occurs for more than 50% of the consumers with the global optimum and the parallel approval method. With the simultaneous approval method, cost increase occurs for all consumers, while no consumer suffers from cost increase of more than 0.1%.

B. Negotiation Results at 65% Relative Group Power Limit

Figure 4(b) shows the results for the load profile negotiation at a group power limit of 4550 kW. While in most time slots the load is similar to the global optimum, larger differences can be seen at 6:00, 8:00, and 15:00. The sequential approval yields increased costs close to the global optimum as shown in Table I. While the increased costs caused by the simultaneous

approval method exceed the optimum by a factor of 7, with approximately 1% they are still very low. However, according to Figure 5(b) the simultaneous approval method does not only lead to the highest cost increase but also to the most uneven distribution of the cost increase among the consumers.

C. Negotiation Results at 55% Relative Group Power Limit

The results for the load profile negotiation at a group power limit of 3850 kW are shown in Figure 4(c). The low group power limit compared to the total energy demand forces the consumers to shift more energy demand to the secondary business hours. Due to the additional costs, this leads to higher total costs. In Table I we can see that even the global optimum leads to an increase of approximately 4% compared to the preferred load profile of each consumer. The sequential approval method leads to an increase of 6%, and the simultaneous approval leads to an increase of approximately 12%. Figure 5(c) does not show a significant difference regarding the evenness of the distribution of the cost increase.

D. Scheduling Overhead

Table II shows the average number of load profiles that a consumer computes before the violation of the group power limit is resolved. The sequential approval method requires the computation of slightly less load profiles compared to the simultaneous approval method.

TABLE II: Average number of load scheduling cycles per consumer.

	Group power limit		
	5950 kW (85%)	4550 kW (65%)	3850 kW (55%)
Sequential approval	17	53	90
Simultaneous approval	18	63	122

VII. Conclusion

Optimized load scheduling based on day-ahead energy price forecasts may lead to demand peaks that cannot be satisfied due to grid limitations. In this paper, we proposed approaches for load profile negotiation that do not require knowledge of internal enterprise details at the aggregator. The results for the given scenario are close to the optimum computed using global knowledge. For lower group power limits compared to the sum of all individual power limits, the sequential approval method yields a lower increase of total costs compared to the simultaneous approval method.

Due to the simplified model, the results cannot be generalized. However, the results show that it is possible to use load profile negotiation to comply with power limits in a day-ahead price optimization scenario. The cost increase is higher compared to a central optimization using global knowledge, but except for very low group power limits (see Section VI-C) the total cost increase is quite small.

The first-come-first-serve property of the sequential approval method leads to penalties for late submissions and can be considered unfair. However, the expectation that the simultaneous approval method leads to a more even distribution of cost increase does not hold for low group power limits. Additionally, for the simultaneous approval method an incentive for submitting the requested number of different load profiles and a distance metric to quantify the degree of difference between submitted load profiles are required.

Opportunities for future research include investigations with more complex mechanisms and more elaborated consumer models.

Acknowledgment

The research leading to these results has received funding from the German Federal Ministry for Economic Affairs and Energy under the ZIM programme (Zentrales Innovationsprogramm Mittelstand), grant no. 16KN039521. The authors alone are responsible for the content of this paper.

References

[1] F. Heimgaertner, U. Ziegler, B. Thomas, and M. Menth, "A Distributed Control Architecture for a Loosely Coupled Virtual Power Plant," in *ICE/IEEE International Technology Management Conference (ICE/IEEE ITMC)*, Jun. 2018.

[2] C. Ibars, M. Navarro, and L. Giupponi, "Distributed Demand Management in Smart Grid with a Congestion Game," in *IEEE International Conference on Smart Grid Communications (SmartGridComm)*, 2010, pp. 495–500.

[3] B. Biegel, P. Andersen, J. Stoustrup, and J. Bendtsen, "Congestion Management in a Smart Grid via Shadow Prices," in *8th Power Plant and Power System Control Symposium (PPPSC)*, Sep. 2012.

[4] S. Huang, Q. Wu, H. Zhao, and C. Li, "Distributed Optimization based Dynamic Tariff for Congestion Management in Distribution Networks," *IEEE Transactions on Smart Grid*, vol. 10, no. 1, pp. 184–192, 2019.

[5] S. Huang and Q. Wu, "Real-Time Congestion Management in Distribution Networks by Flexible Demand Swap," *IEEE Transactions on Smart Grid*, vol. 9, no. 5, 2018.

[6] K. G. Boroojeni, M. H. Amini, S. S. Iyengar, M. Rahmani, and P. M. Pardalos, "An Economic Dispatch Algorithm for Congestion Management of Smart Power Networks," *Energy Systems*, vol. 8, no. 3, pp. 643–667, 2017.

[7] J. Bagemihl, F. Boesner, J. Riesinger, M. Künzli, G. Wilke, G. Binder, H. Wache, D. Laager, J. Breit, M. Wurzinger, J. Zapata, S. Ulli-Beer, V. Layec, T. Stadler, and F. Stabauer, "A Market-Based Smart Grid Approach to Increasing Power Grid Capacity Without Physical Grid Expansion," *Computer Science - Research and Development*, vol. 33, no. 1, pp. 177–183, 2018.

[8] J. Hazra, K. Das, and D. P. Seetharam, "Smart Grid Congestion Management Through Demand Response," in *IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2012, pp. 109–114.

[9] O. Sundstrom and C. Binding, "Flexible Charging Optimization for Electric Vehicles Considering Distribution Grid Constraints," *IEEE Transactions on Smart Grid*, vol. 3, no. 1, pp. 26–37, 2012.