Optimization algorithms for the scheduling of IEEE 802.1 Time-Sensitive Networking (TSN)

Michael Lander Raagaard   Paul Pop

January 2017

DTU Technical Report

DTU Compute
Technical University of Denmark
2800 Kongens Lyngby, Denmark

Abstract

TSN is an IEEE effort to bring deterministic real-time capabilities to IEEE 802.3 Ethernet. TSN meets the bandwidth and dependability requirements of emerging mixed-criticality applications, while ensuring timeliness of time-critical communication. The IEEE 802.1Qbv substandard introduces time-aware gates within network devices enabling fully deterministic temporal behavior of real-time communication. For each egress port, a Gate Control List (GCL) specifies which queue (traffic class) may transmit at which points in time. Using this functionality enables frames to be forwarded in the network in a time-triggered manner. In this report, we are interested in synthesizing schedule tables for periodic time-critical flows on TSN architectures such that timeliness is guaranteed, i.e., an assignment of critical Ethernet frames to egress port queues, and a construction of GCLs ensuring that all frames are schedulable. The problem is formulated as a multi-objective combinatorial optimization problem, which minimizes the queue utilization as well as end-to-end latency. We propose several algorithms to solve this problem, and evaluate them on synthetic and realistic test cases.

Abbreviations

Abbreviation	**Description**
ALAP	as-late-as-possible
ASAP	as-soon-as-possible
AVB	Audio-Video Bridging
BE	Best-Effort
CBS	Credit Based Shaper
FIFO	first in, first out
GCL	Gate-Control List
GRASP	Greedy Randomized Adaptive Search Procedure
ILP	Integer Linear Programming
IQR	Interquartile Range
LCM	Least Common Multiple
MIP	Mixed Integer Programming
MTU	Maximum Transmission Unit
OMT	Optimization Modulo Theories
PCP	Priority Code Point
QoS	Quality of Service
RCL	Restricted Candidate List
SMT	Satisfiability Modulo Theories
TAS	Time Aware Shaper
TSN	Time-Sensitive Networking
TT	Time-Triggered
VID	VLAN Identifier
WCD	Worst-Case Delay

Notations

Notation	**Description**
A	Set of repetitions of s_i within hyperperiod
α	Individual repetition of s_i within hyperperiod
a	Local search strategy
B	Set of repetitions of s_j within hyperperiod
β	Individual repetition of s_j within hyperperiod
δ	Maximum synchronization error
$D_\pi(x)$	Combinations of choosing π link-sharing flows from x
$\mathbf{E}$	Data links in network

Notation	**Description**
ϵ	Variable for modeling identical queue assignments
$\mathbf{ES}$	End systems in network
$\mathcal{F}$	Set of all frames
$f(x)$	Metaheuristic objective function
$f_{i,m}^{[v_a,v_b]}$	A single frame
$f_{i,m}^{[v_a,v_b]}.L$	Transmission duration of $f_{i,m}^{[v_a,v_b]}$
$f_{i,m}^{[v_a,v_b]}.\phi$	Periodic offset of $f_{i,m}^{[v_a,v_b]}$
$f_{i,m}^{[v_a,v_b]}.\underline{\phi}$	Lower bound on periodic offset
$f_{i,m}^{[v_a,v_b]}.\overline{\phi}$	Upper bound on periodic offset
γ	Length of Restricted Candidate List (RCL)
$\mathcal{G}(\mathbf{E}, \mathbf{V})$	Directed graph of network topology
$\mathcal{I}$	Set of all flow instances
K	Excess queue usage
$\kappa_{a,b}$	Number of queues used by TT flows in $[v_a, v_b]$
Λ	Additional end-to-end latency
l	Ratio of time spent in local search phase
λ_i	End-to-end latency for s_i
$\underline{\lambda}_i$	Lower bound on end-to-end latency
M	Theoretically infinitely large constant
ω	Variable for modeling disjunction
π	Maximum number of flows to destroy
Φ_α	Feasible region in repetition α
$\Phi_\cap$	Feasible regions intersection
$\underline{\phi}$	Function for lower bound on period offsets
$\mathcal{R}$	Set of all frame repetitions
$R_{a,b}$	Set of all routes from v_a to v_b
r_k	Route in network
$\mathcal{S}$	Set of all flows
σ	Variable for modeling disjunction
s_i	A single flow
$s_i.D$	Deadline of s_i
$s_i.I$	Set of flow instances associated with s_i
$s_i.k$	Number of frames on each link of s_i
$s_i.r$	Route of s_i
$s_i.s$	Sending link of s_i
$s_i.T$	Period of s_i
$s_i.t$	Receiving link of s_i
$s_i.v_a$	Sending end system for s_i

Notation	**Description**
$s_i^{[v_a,v_b]}$	Instance of flow s_i on link $[v_a, v_b]$
$s_i^{[v_a,v_b]}.F$	Frames associated with $s_i^{[v_a,v_b]}$
$s_i^{[v_a,v_b]}.\rho$	Queue assignment for $s_i^{[v_a,v_b]}$
$s_i.v_b$	Receiving end system for s_i
$\mathbf{SW}$	Switches in network
$\mathbf{V}$	Devices in network
v_a	Network device, either switch or end system
$[v_a, v_b]$	Data link from v_a to v_b
$[v_a, v_b].c$	Number of queues associated with egress port $[v_a, v_b]$
$[v_a, v_b].d$	Propagation delay on link $[v_a, v_b]$
$[v_a, v_b].s$	Transmission rate on link $[v_a, v_b]$
X	Solution space
x	Feasible solution
x^*	Optimal solution
z	Objective value
z^*	Optimal objective value

1 Introduction

A *safety-critical system* is a system whose failure or malfunction may cause serious harm to humans, equipment, or the environment. For example, nuclear plant control, aerospace, or automotive systems. Safety-critical systems usually run distributed *real-time* applications with nodes communicating through a network. In addition to the computational value, the correctness of a real-time application depends on the time at which results are produced [1]. In other words, real-time systems must guarantee response within specified time constraints. Consequently, the communication network must meet certain timeliness requirements to ensure correctness of the application, such as guaranteeing that messages are delivered within their deadlines. In this report, we model and solve combinatorial optimization problems related to communication networks for distributed real-time safety-critical systems.

Many examples of real-time safety-critical systems exist within the domains of industrial automation and automotive systems. For instance, consider an automated manufacturing plant with robots working on production lines [2, 3]. Each robot is made up of several end systems, such as sensors, controllers, and actuators. For cooperation purposes, the robots are interconnected in a physical network. In addition, the robots are connected to a centralized control unit with more computational power than available in each robot. Physical processes, such as a system of robots, are highly time-sensitive. Missed deadlines in this context could lead to damaged equipment or even endanger the lives of workers near the production line. Consider for instance that a sensor in a robot detects a value that requires all robots to stop immediately. The sensor value is transmitted through the network to the centralized control unit which decides to make a stop. Now, it sends a stop message to each robot, which signals the corresponding actuator. This scenario requires a reliable network with timing guarantees for critical traffic.

Traditionally, fieldbus [4] technologies, such as CAN [5] and FlexRay [6], have been used for industrial real-time applications. However, emerging applications have increasing bandwidth demands. For instance, autonomous driving requires data rates of at least 100 Mbit/s for graphical computing based on camera, radar, and lidar data, whereas CAN and FlexRay only provide data rates of up to 0.5 Mbit/s and 10 Mbit/s, respectively [7].

The well-known networking standard *IEEE 802.3 Ethernet* [8] meets the emerg-

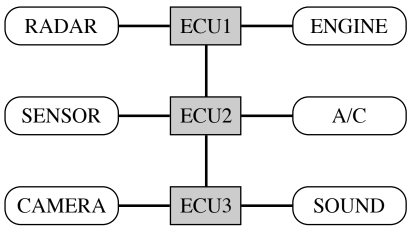


Figure 1: Safety-critical in-vehicle communication network for the automotive domain.

ing bandwidth requirements for safety-critical networks, while remaining scalable and cost-effective. It does, however, lack real-time capabilities [9]. Hence, there is a need for Industrial Ethernet [10], i.e., Ethernet in an industrial environment extended with deterministic real-time capabilities. Many extensions, such as EtherCAT [11], PROFINET [12], SERCOS III [13], and TTEthernet [14], have previously been suggested and used in the industry. Although they satisfy the timing requirements, they are incompatible with each other, and as a result, they cannot operate on the same physical links in a network without losing real-time guarantees [3]. Consequently, the *IEEE 802.1 Time-Sensitive Networking (TSN)* task group [15] has since 2012 worked on standardizing real-time and safety-critical enhancements for Ethernet.

The work in this report is based on networks as defined in the Time-Sensitive Networking (TSN) standard. In such networks, applications with different timeliness requirements coexist. Network traffic is divided into three categories based on criticality level: (1) Hard real-time communication where missing a deadline results in failure, (2) soft real-time communication where deadline misses are undesired but safe, and (3) best-effort communication without timing requirements.

Fig. 1 shows a simplified example from the automotive domain of a network used for all three traffic types. The figure depicts an in-vehicle network [16, 17] of modern cars. In this type of network, hundreds of electronic control units (ECUs) are interconnected for running applications with different timing requirements. For instance, applications for driver assistance systems and autonomous driving require hard real-time communication, while others, e.g., related to on-board entertainment or driver's comfort have soft real-time requirements or none at all. In the network of Fig. 1, autonomous driving would rely on data from cameras, sensors, and radars to avoid collisions, while an entertainment application streams data to the sound system. It is crucial that streaming

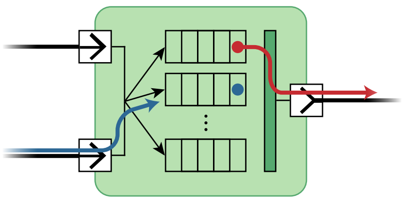


Figure 2: Ethernet switch with frames waiting in queues.

to the sound system does not affect timeliness of the collision detection. The in-vehicle network must meet the real-time requirements of all applications.

In Ethernet networks, messages are transmitted between end systems as *frames*. Frames are forwarded on links, through switches, on a route from sender to receiver. They queue up in switches during transmission while waiting for the next link in the route to become available. Each switch has multiple queues, and frames are filtered into queues based on their priority. When a link becomes available a new frame is chosen for transmission starting from the top queue. Hence, the queueing delay for each frame depends on its priority, on how many other frames are queued in front of it, and on the availability of the next link. This leads to network congestion causing nondeterministic behavior.

Fig. 2 illustrates an Ethernet switch with its internal queues. The switch has two incoming links on the left and one outgoing link on the right. One red and one blue frame are queued in the first and second queue, respectively. The red frame is chosen for transmission on the outgoing link because it is queued in the topmost queue. Meanwhile, the blue frame waits in the second queue for the link to become available. A second blue frame is arriving at the lower incoming link and is queued in the second queue with the other blue frame.

TSN specifies three traffic classes, denoted Time-Triggered (TT), Audio-Video Bridging (AVB), and Best-Effort (BE). Time-Triggered (TT) traffic is used for hard real-time applications where non-determinism is unacceptable. TSN defines mechanisms for forwarding queued frames from a specific queue at precise points in time. This is implemented by blocking the other queues and relies on static schedule tables for deciding when to block each queue. These mechanisms provide the basic building blocks for achieving determinism and bounded end-to-end latency. It is, however, beyond the scope of TSN to define how to construct the schedule tables.

The second traffic class, Audio-Video Bridging (AVB), is used for soft real-time applications such as multi-media streaming, where end-to-end latency and jitter should be minimized in general, but an occasional deadline miss is acceptable. Best-Effort (BE) traffic is — as the name suggests — suitable for best-effort traffic. It is only eligible for transmission when it does not interfere with TT and AVB traffic. Hence, it is to be used for low priority traffic without timing requirements.

In this report, we focus on the scheduling problem for TT traffic. We assume that the TSN network topology and TT messages are given as input. The objective of the report is to design algorithms for synthesizing schedule tables for TT traffic in order to guarantee timeliness. The schedule must ensure that all deadlines are satisfied. Furthermore, it should be optimized to meet industrial application demands [18, 19] for minimal latency and jitter. The schedule must facilitate a high Quality of Service (QoS) for AVB and BE traffic by using a minimal number of queues in switches.

We propose three different approaches: (1) One that finds optimal schedules based on an Integer Linear Programming (ILP) formulation. (2) A scalable, constructive heuristic for finding feasible schedules. (3) A Greedy Randomized Adaptive Search Procedure (GRASP)-based heuristic as a trade-off between tractability and solution quality yielding high-quality schedules in reasonable time. For comparison, the three approaches are experimentally evaluated on industrial-sized test cases with high link utilization.

2 Related Work

Synthesizing schedules for distributed real-time applications is a well-studied problem. It consists of two coupled problems: Scheduling tasks at the processor level and scheduling frames at the network level. Real-time applications rely on timeliness at both levels. In [19] the two problems are solved jointly using a Satisfiability Modulo Theories (SMT) approach, and Zhang et al. [20] propose a Mixed Integer Programming (MIP) approach for the joint problem.

In this report, however, we focus on synthesizing schedules at the network level. In the context of TTEthernet, numerous strategies have been proposed for scheduling frames on network links. Steiner [21] models the problem as a set of constraints which are then solved by an SMT-solver. In [22] the same author proposes an extension to the SMT approach where blank spaces are left between TT frames to avoid starvation of lower-priority frames, thereby improving QoS. Pozo et al. [23] improve scalability of the SMT-based approach by decomposing the problem into subproblems which are solved independently. Doing that, they are able to find feasible schedules for up to 100,000 frames in one hour.

Other approaches have also been proposed, such as a Tabu Search-based meta-heuristic [24] which minimizes end-to-end latency of lower priority traffic. Avni et al. [25] propose a scheduling strategy where switches have an error-recovery protocol, enabling them to utilize secondary paths in case of link failures.

Recent work has addressed the scheduling problem for TSN as well. TSN and TTEthernet are similar in many ways, but differ in some significant aspects: Messages in TTEthernet consist of a single frame, whereas TSN messages may consist of multiple frames. Furthermore, TTEthernet schedule tables are specified for individual frames, where TSN specifies schedules on a per-queue basis. Consequently, all messages sharing the same queue are affected by the associated schedule table. As a result, the work on TTEthernet scheduling is not directly applicable to TSN networks.

Dürr and Nayak [3] relate the TSN scheduling problem to the “No-wait Job-shop Scheduling Problem” in order to achieve schedules with minimum network delay for TT messages. They assume that a single queue is reserved for TT traffic in every outgoing port and that all messages are repeatedly transmitted within the same period. With these assumptions they achieve near-optimal schedules with an Tabu Search-based approach.

Craciunas et al. [26] identify issues affecting determinism in TSN schedules and define additional TSN-specific constraints for overcoming the issues. Furthermore, they propose an SMT-based approach where the assumptions from [3] have been relaxed, i.e., TT traffic may utilize multiple queues, and messages have individual periods. Assigning TT traffic to multiple queues adds flexibility and enables finding feasible schedules in scenarios with high link utilization. However, queue usage should in general be minimized where possible in order to make queues available for lower-priority traffic. Thus, [26] also presents an Optimization Modulo Theories (OMT) approach for minimizing queue usage. The work in this report considers the same constraints and assumptions as in [26] and is an extension of the work we have presented in [27].

3 Report Overview

The report is structured into nine sections as follows:

Section 2 introduces fundamental Ethernet concepts and describes relevant TSN protocols.

Section 3 presents the abstraction models for the architecture and application, i.e., the network and communication models.

Section 4 gives a formal definition of the problem and a motivational example.

Section 5 describes an ILP-based approach for solving the problem to optimality.

Section 6 proposes a constructive heuristic that finds feasible solutions in most cases.

Section 7 presents an optimization layer on top of the heuristic approach. The layer is a metaheuristic which attempts to improve solution quality.

Section 8 experimentally evaluates and compares the three approaches on synthetic test cases. They are evaluated in terms of solution quality and execution time.

Section 9 concludes the report and describes ideas for future work.

4 Time-Sensitive Networking

This section gives a brief introduction to Time-Sensitive Networking (TSN). TSN defines three traffic classes: Time-Triggered (TT), Audio-Video Bridging (AVB), and Best-Effort (BE). The report considers the TT scheduling problem, and hence, the introduction will primarily focus on TT traffic. At the time of writing, the work on TSN is still ongoing. In this report we assume TSN consisting of the standards presented below.

IEEE 802.1ASrev	Timing and Synchronization [28]
IEEE 802.1Qav	Forwarding and Queueing of Time-Sensitive Streams [29]
IEEE 802.1Qbv	Enhancements for Scheduled Traffic [30]
IEEE 802.1Qbu	Frame Preemption [31]

Some fundamental Ethernet concepts are introduced before the standards are explained and related to the traffic classes.

5 Ethernet Networking

TSN is based on a switched multi-hop network architecture adopted from *IEEE 802.3 Ethernet*. Switches interconnect end systems via *full-duplex* links, meaning that the physical links enable transmission in both directions simultaneously. Fig. 3 shows a TSN network architecture with end systems ES_1, ES_2, ES_3, ES_4, and ES_5 interconnected via network switches SW_1 and SW_2. In the figure, a message is sent from ES_2 to ES_4 via SW_1 and SW_2.

5.1 Frames

Messages sent on Ethernet networks are wrapped in Ethernet frames [8, 32]. A single frame transmits a payload of at most 1500 bytes, the so-called Maximum

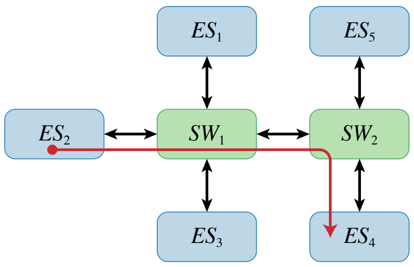


Figure 3: Switched multi-hop full-duplex TSN network.

Transmission Unit (MTU). If the data size of a message is larger than MTU, it is fragmented into multiple frames. The number of frames needed to transmit a message with data size x is calculated as $\lceil \frac{x}{MTU} \rceil$. All frames are assumed to be MTU-sized, except for the last which contains the remaining data.

In the physical layer, an Ethernet frame itself is wrapped in an Ethernet packet [4, 8]. Fig. 4 shows the different fields in Ethernet packets and frames. The packet contains a preamble (7 bytes), a start frame delimiter (SFD) (1 byte), the frame (64–1522 bytes), and an inter frame gap (12 bytes). The purpose of the preamble (including SFD) and the inter frame gap is to mark the beginning and end of a new incoming frame. The frame itself consists of six fields. The first two (each 6 bytes) contain MAC addresses of the receiver and sender, respectively. Then comes the *IEEE 802.1Q* header [32] (4 bytes), the length of the frame (2 bytes), followed by the payload or message data (42–1500 bytes). The final field is the frame check sequence (4 bytes) which is a checksum for detecting if the frame was corrupted during transmission.

Two fields within the *IEEE 802.1Q* header are of importance to TT traffic:

(i) The VLAN Identifier (VID) is a 12-bit field specifying the Virtual LAN of a frame. This is used to distinguish frames from different messages.

(ii) The Priority Code Point (PCP) is a 3-bit field specifying the priority level, i.e., the traffic class such as TT, AVB, or BE. Furthermore, it defines which queue the frame is assigned to within a switch.

A total overhead of 42 bytes is added to every payload sent on the network. If the payload is less than the minimum size of 42 bytes then it is padded with

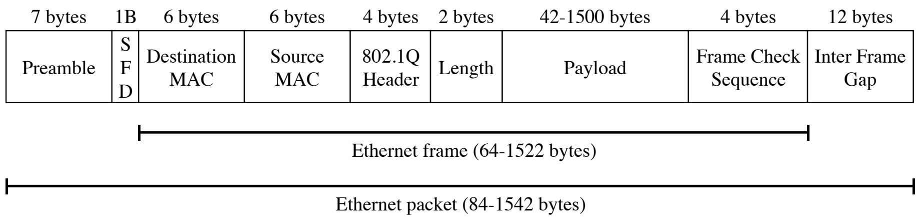


Figure 4: Overview of Ethernet frame fields.

zeros until reaching the minimum size. Thus, in this case the overhead is even larger.

5.2 Switches

An Ethernet switch has ingress (incoming) and egress (outgoing) ports connecting it via links to surrounding switches and end systems. Each egress port typically has eight queues for storing frames that wait to be forwarded on the corresponding link, as shown in Fig. 2. When frames arrive at ingress ports they are filtered into queues based on their PCP field in the *IEEE 802.1Q* header. When a link is idle, a new frame is chosen for transmission from among the queued frames. If frames are waiting in multiple queues, the topmost queue is selected, and hence the queue assignment is related to the priority of the frame. If there are always frames queued in a high-priority queue, this selection scheme leads to starvation of lower-priority frames. The AVB traffic class is an effort to prevent this, while still providing low latencies and minimal jitter for the high-priority frames. It is described in Sect. 6.

6 AVB and BE Traffic

AVB traffic as defined in *IEEE 802.1Qav* is suitable for soft real-time communication such as multi-media streaming. In such applications latency and jitter should be minimized. AVB is implemented using a Credit Based Shaper (CBS) for shaping traffic. There is one CBS for each AVB-enabled queue in the switch. Frames are available for transmission when the credit is non-negative. As AVB frames are transmitted, the credit decreases with a *send slope.* Conversely, the credit increases with an *idle slope* slope when AVB frames are waiting in the queue. The purpose of CBS is to even out AVB traffic to prevent starvation of lower priority messages [27]. However, the credit based shaper is not robust and can cause congestion loss [33], which makes it unsuitable for hard real-time communication.

BE traffic is suitable for low-priority traffic, where there are no timing requirements. Hence, BE frames are eligible for transmission when the link is idle, and there are no eligible frames in higher-priority queues.

In this report, we do not explicitly consider AVB and BE traffic because they do not influence the TT schedule. Instead, the TT schedule affects the QoS of AVB and BE in the following way: A schedule that takes up many queues for the TT traffic, leaves fewer queues available for AVB and BE traffic. Consequently, schedules using less queues are preferable, as they leave more queues available for AVB and BE traffic.

7 Time-Triggered (TT) Traffic

TSN defines standards that bring real-time capabilities to Ethernet in the form of TT traffic. *IEEE 802.1ASrev* defines a network-wide time-synchronization protocol which effectively achieves a global notion of time across all switches and end systems. Clocks must be comparable across devices, if forwarding of frames is to be controlled accurately. The time-synchronization protocol achieves a microsecond network precision, denoted by δ. The difference in the internal clocks of any two devices at any point in time is at most δ.

Utilizing the global network clock, *IEEE 802.1Qbv* defines a Time Aware Shaper (TAS) concept which enables scheduling of high priority traffic. TAS controls a gate for each queue. Frames in a queue are only eligible for transmission if the queue gate is open. Fig. 5 shows a network diagram with three end systems, ES_1, ES_2, ES_3, and a single switch, SW_1. The switch has two ingress ports and a single egress port. Egress ports are depicted as boxed arrow tails, ingress ports as boxed arrow heads, and data links as the lines in-between. Every queue of the egress port has an associated gate. The gate of the topmost queue is opened,

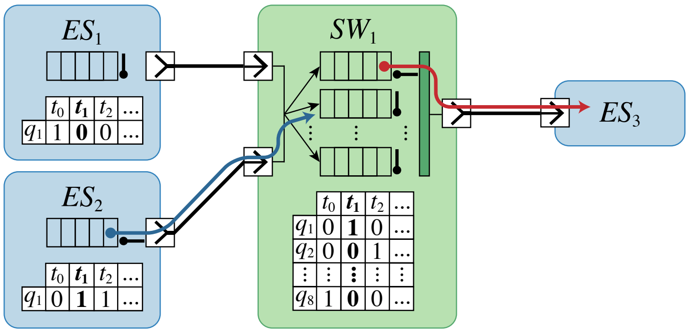


Figure 5: TSN network with diagrams of internal components of end systems and switches. The figure depicts the situation at time t_1 where a frame (blue arrow) is entering switch SW_1 from end system ES_2. In parallel another frame (red arrow) is leaving SW_1 at an egress port towards end system ES_3.

enabled the red frame to be selected for transmission. Interference from lower priority traffic is prevented by closing the gates of the remaining queues, as shown in the figure. When the egress port is idle, the next frame is selected for transmission from the queue with highest priority among the queues with open gates. Opening queues in a mutually-exclusive fashion, allows for full control of frame forwarding.

In this report, we assume that all TSN devices are scheduled. It means that all egress ports have a TAS controlling when traffic is transmitted. Conceptually, when only TT traffic is considered, the egress ports of end systems have a single queue with an associated gate, as shown in the diagrams for ES_1 and ES_2.

A Gate-Control List (GCL) defines for each egress port, when the queue gates are open and closed. In the figure they are depicted as white tables below queues. 1 and 0 in the GCL represent an open and closed gate, respectively. For instance, the gate in ES_1 is open at time t_0 but closed at t_1 and t_2, and conversely for the gate in ES_2. Using the GCLs to schedule forwarding of frames in a route from sender to receiver, enables TT traffic suitable for hard real-time communication.

The GCLs can be constructed in such a way that AVB and BE traffic are prevented from initiating transmission in the time slots reserved for TT frames. However, non-determinism could still occur, if a lower priority frame is already transmitting on the link at the beginning of a time slot. There are two ways to prevent this. The first option is to place an MTU-sized *guard band* before every TT time slot. The guard band closes all other queues well in advance to ensure that the link is available when it is time to transmit the TT frame. This option is undesirable because it decreases the bandwidth available for lower-priority traffic. The second option is to let the TT frames *preempt* to lower priority queues as defined in *IEEE 802.1Qbu*. When a frame is preempted, its transmission is temporarily paused, so the link becomes available to a higher-priority frame. Once transmission of the higher-priority frame completes, transmission of the preempted frame is resumed. A header is added to each fragment of the preempted frame, which increases the total overhead associated with the frame. However, the overhead is negligible compared to an MTU-size guard band. Thus, preemption is the preferred method, if supported by the hardware.

Each GCL has a temporal granularity which we refer to as the *macrotick*. For simplicity, and without loss of generality, we assume that all GCLs have a macrotick of 1 µs. This makes comparing GCLs across different switches straightforward.

Fig. 5 shows the state of the network at time t_1. At this point in time, the first gate in SW_1 is open, while the others are closed, as defined in column t_1 of the GCL. A red frame is queued in q_1 of SW_1, and the link connecting SW_1 and ES_3 is idle. Consequently, the frame in q_1 is chosen for transmission because q_1 is the only queue with an open gate. This is illustrated with the red arrow.

In parallel, a frame, illustrated with a blue arrow, is being transmitted from ES_2 to SW_1 because the gate in ES_2 is open. The PCP field of the frame dictates that it is filtered into q_2 in SW_1. The gate of this queue is closed at time t_1, so it cannot interfere with transmission of the red frame. At time t_2, the red frame has finished transmission. From the t_2-column of the GCL in SW_1, it is seen that q_1 closes and q_2 opens, causing the blue frame to be transmitted to ES_3.

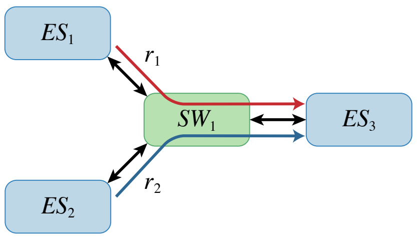


Figure 6: Architecture model of Fig. 5 with vertices representing end systems and switches, and edges representing data links.

8 System Model

This section presents the system model used throughout the report. More specifically, it describes the architecture and application model. The architecture model is an abstract representation of the physical TSN network, including end systems, switches, and links. The application model is an abstract representation of the communication protocol for sending messages on a TSN network. The presented notation is inspired by Craciunas et al in [26].

9 Architecture Model

The topology of a TSN network is modeled as a directed graph $\mathcal{G}(\mathbf{E}, \mathbf{V})$, where the set of vertices $\mathbf{V}$ is the devices in the network, i.e., the set of all end systems, denoted $\mathbf{ES}$, and the set of all switches, denoted $\mathbf{SW}$. Hence, $\mathbf{V} = \mathbf{ES} \cup \mathbf{SW}$. The set of edges $\mathbf{E}$ represents data links in the network. A directed edge from $v_a \in \mathbf{V}$ to $v_b \in \mathbf{V}$ represents a one-way communication link from v_a to v_b. Thus, a physical full-duplex link between devices v_a and v_b results in two edges, denoted $[v_a, v_b] \in \mathbf{E}$ and $[v_b, v_a] \in \mathbf{E}$. Fig. 6 shows the architecture model of the network in Fig. 5.

Associated with each link is the tuple of attributes (s, d, c) which is explained in Table 3. Throughout the report an object-oriented notation is used to refer to specific attributes. E.g., $[v_a, v_b].s$ refers to the transmission rate of link $[v_a, v_b]$. The transmission rate is typically 100 Mbps or 1 Gbps. The propagation delay

$s \in \mathbb{R}$	Transmission rate
$d \in \mathbb{R}$	Propagation delay
$c \in \mathbb{N}$	Number of queues in corresponding egress port

Table 3: Attributes for data link $[v_a, v_b] \in \mathbf{E}$.

of a link is proportional to the length of the physical link. There is exactly one egress port for every link and, hence, the notation $[v_a, v_b]$ refers to the link from v_a to v_b as well as the egress port in v_a associated with the link to v_b. It should be clear from the context when the notation refers to the link and when it refers to the egress port. We use $[v_a, v_b].c$ to refer to the number of queues in egress port $[v_a, v_b]$. We assume $[v_a, v_b].c = 1$ if $v_a \in \mathbf{ES}$ and $[v_a, v_b].c = 8$ if $v_a \in \mathbf{SW}$, as shown in Fig. 5.

A route, r_k, is an ordered sequence of links connecting sending end system v_a with receiving end system v_b. The sequence represents a data path for sending messages from v_a to v_b. Every route starts and ends in end systems with one or multiple intermediate switch vertices. Fig. 6 shows two routes: r_1 going from ES_1 to ES_3, i.e., $r_1 = ([ES_1, SW_1], [SW_1, ES_3])$, and route $r_2 = ([ES_2, SW_1], [SW_1, ES_3])$. Note that there can be multiple routes connecting the same two end systems depending on the network topology. We denote $R_{a,b}$ the set of all routes connecting v_a with v_b. Due to the simplicity of the topology in Fig. 6 all pair of end systems have only one route, e.g., $R_{1,3} = \{r_1\}$ and $R_{2,3} = \{r_2\}$. In this report we assume all messages are unicast, and consequently every route is a path with a single destination.

10 Application Model

As the focus of this report is on Time-Triggered (TT) traffic, the application model considers only this type of traffic. Hence, AVB and BE is not modeled. The considered TT messages are periodic, which means that they are repeatedly sent along the network with a certain period. Such a periodic message sent on a route in the network is called a *flow*.

$T \in \mathbb{N}$	Period in microseconds
$D \in \mathbb{N}$	Maximum end-to-end latency in microseconds
$v_a \in \mathbf{ES}$	Sending end system (source)
$v_b \in \mathbf{ES}$	Receiving end system (destination)
$r \in R_{a,b}$	A route in the network connecting v_a and v_b
$s \in r$	Sending link, i.e., outgoing link from v_a
$t \in r$	Receiving link, i.e., incoming link in v_b
$k \subset \mathbb{N}$	Number of frames on each link
$I \subset \mathcal{I}$	Set of flow instances

Table 4: Attributes for flow $s_i = (T, D, v_a, v_b, r, s, t, k, I)$.

$\rho \in \{1, \ldots, [v_a, v_b].c\}$	Assigned queue for egress port $[v_a, v_b]$
$F \subset \mathcal{F}$	Set of frames of s_i transmitted on $[v_a, v_b]$

Table 5: Attributes for flow instance $s_i^{[v_a,v_b]} = (\rho, F)$.

10.1 Flow Model

TT communication is expressed as a set of flows $\mathcal{S}$. Associated with each TT flow $s_i \in \mathcal{S}$ is a set of attributes specified in Table 4. The meaning of the attributes for flow s_i is the following: Every $s_i.T$ microseconds, a message, in the form of a sequence of $s_i.k$ frames, is sent on the network via a route $s_i.r$ from end system $s_i.v_a$ to end system $s_i.v_b$. In each period, the entire message must be received in $s_i.v_b$ within $s_i.D$ microseconds from the time the transmission is initiated in $s_i.v_a$. In other words, the last frame in the sequence must be fully received within the deadline.

A flow *instance*, denoted $s_i^{[v_a,v_b]} \in \mathcal{I}$, is the instance of a flow s_i on a particular link $[v_a, v_b]$. $\mathcal{I}$ denotes all flow instances in a schedule, and $s_i.I$ refers to all flow instances of s_i. There is one such flow instance for each link in the route $s_i.r$. A tuple (ρ, F) is associated with each flow instance and the attributes are described in Table 5. The flow instance concept is introduced to denote the set of frames $s_i.F$ transmitted on the individual links for flow s_i. It is also introduced to capture the assignment of frames to egress port queues. Recall that $[v_a, v_b]$ denotes the link between devices v_a and v_b as well as the egress port in v_a towards v_b. The frames $s_i^{[v_a,v_b]}.F$ are queued in egress port $[v_a, v_b]$ during transmission. The attribute $s_i^{[v_a,v_b]}.\rho$ specifies to which queue the frames are assigned, thus it is upper bounded by the number of queues in the device ($[v_a, v_b].c$). The flow instance concept ensures that all frames in $s_i^{[v_a,v_b]}.F$ are

$L \in \mathbb{N}$	Transmission duration in microseconds on $[v_a, v_b]$
$\phi \in [0, s_i.T - L]$	Integral microsecond offset within each period

Table 6: Attributes for frame $f_{i,m}^{[v_a,v_b]} = (L, \phi)$.

assigned the same queue, while allowing frames of flow s_i to be assigned different queues in different egress ports along route $s_i.r$.

10.2 Frame Model

We let $\mathcal{F}$ denote the set of all frames in a schedule. A single frame $f_{i,m}^{[v_a,v_b]} \in \mathcal{F}$ denotes the transmission of the mth frame of flow s_i on link $[v_a, v_b]$. Frame $f_{i,m}^{[v_a,v_b]}$ is associated with the attribute pair (L, ϕ) as described in Table 6. The transmission duration $f_{i,m}^{[v_a,v_b]}.L$ is based on the transmission rate and propagation delay of the physical link $[v_a, v_b]$, and the total number of bytes in the packet. Suppose that the frame has a payload of MTU (1500 bytes), resulting in a packet size of 1542 bytes, then the transmission duration on a 1 Gbps link with negligible propagation delay is calculated in Eq. 1.

$$f_{i,m}^{[v_a,v_b]}.L = \frac{1542 \text{ bytes}}{[v_a, v_b].s} + [v_a, v_b].d = \frac{1542 \text{ bytes}}{1 \text{ Gbps}} = 12.336\,\mu\text{s} \tag{1}$$

By only considering the transmission duration of a particular frame, we can ignore the transmission rate and propagation delay of individual links.

The offset $f_{i,m}^{[v_a,v_b]}.\phi$ defines the start time for the frame transmission within each period $s_i.T$. The frame is repeatedly sent at times

$$\phi, \quad s_i.T + \phi, \quad 2 \cdot s_i.T + \phi, \quad 3 \cdot s_i.T + \phi, \quad \ldots$$

For simplicity, we restrict that every frame must be fully transmitted within its period, yielding a feasible offset interval of $\phi \in [0, s_i.T - L]$. Consequently, the end-to-end latency of any flow is upper bounded by its period, which means that only a deadline value less than the period is meaningful ($s_i.D \leq s_i.T$).

11 Gate-Control List (GCL) Schedule

GCLs for all egress ports make up a deterministic schedule of when to send TT frames on links. Because of the periodic nature of flows, the GCLs have a certain cycle time after which they start over from the beginning. Consequently, the corresponding schedule repeats after this cycle duration which we denote the *hyperperiod* [1]. The hyperperiod depends on the periods of the individual flows because the period of each flow must be a divisor of the hyperperiod, i.e., every flow period repeats an integral number of times within the hyperperiod. Hence, the hyperperiod is the Least Common Multiple (LCM) of all the flow periods.

To illustrate the correspondence between GCLs and the resulting schedule, as well as the newly introduced notation, we present an example: Two flows s_1 and s_2 are given in the architecture from Fig. 6 on routes r_1 and r_2, respectively. We assume that all data links have a transmission rate of 1 Gbps and negligible propagation delay. Flow s_1 has a data sizes of 1500 bytes, and flow s_2 a data size of 4500 bytes, resulting in one and three MTU-sized frames, respectively. All frames have equal duration of 12.336 µs as calculated in Eq. 1. The periods of s_1 and s_2 are 100 µs and 150 µs, respectively. The flows are given by the tuples:

$$
\begin{aligned}
s_1 = (&100\,\mu\text{s}, 100\,\mu\text{s}, ES_1, ES_3, r_1, [ES_1, SW_1], [SW_1, ES_3], && (2)\\
&1, \{s_1^{[ES_1,SW_1]}, s_1^{[SW_1,ES_3]}\}) && (3)\\
s_2 = (&150\,\mu\text{s}, 150\,\mu\text{s}, ES_2, ES_3, r_2, [ES_2, SW_1], [SW_1, ES_3], && (4)\\
&3, \{s_2^{[ES_2,SW_1]}, s_2^{[SW_1,ES_3]}\}) && (5)
\end{aligned}
$$

The individual flow instances are defined as follows:

$$
\begin{aligned}
s_1^{[ES_1,SW_1]} &= (1, \{f_{1,1}^{[ES_1,SW_1]}\}) && (6)\\
s_1^{[SW_1,ES_3]} &= (1, \{f_{1,1}^{[SW_1,ES_3]}\}) && (7)\\
s_2^{[ES_2,SW_1]} &= (1, \{f_{2,1}^{[ES_2,SW_1]}, f_{2,2}^{[ES_2,SW_1]}, f_{2,3}^{[ES_2,SW_1]}\}) && (8)\\
s_2^{[SW_1,ES_3]} &= (2, \{f_{2,1}^{[SW_1,ES_3]}, f_{2,2}^{[SW_1,ES_3]}, f_{2,3}^{[SW_1,ES_3]}\}) && (9)
\end{aligned}
$$

Fig. 7 shows a schedule for transmitting the frames of s_1 and s_2 illustrated as a variant of a Gantt chart [34]. The horizontal axis represents the time dimension, that is, when frames are transmitted. The vertical axis represents links and queues. A box represents the transmission of a frame on a link. For instance, the box labeled "1.1" on the row for $[ES_1, SW_1]$ represents transmission

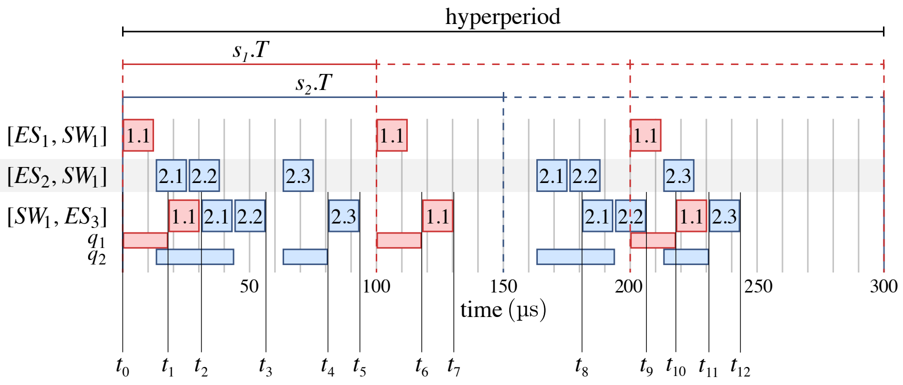


Figure 7: Schedule of frame transmissions on links. Flows s_1 (red) and s_2 (blue) have periods 100 µs and 150 µs, respectively, with hyperperiod 300 µs. The events $t_1, \ldots, t_{12}$ represent changes in GCL for egress port $[SW_1, ES_3]$ (see Table 7).

of $f_{1,1}^{[ES_1,SW_1]}$. The thin rows below link $[SW_1, ES_3]$ illustrate when frames are in the queues of egress port $[SW_1, ES_3]$. A frame is in a queue from the time transmission is initiated on the previous link, until the time transmission is initiated on the egress port associated with the queue. Hence, in Fig. 7, $f_{1,1}^{[ES_1,SW_1]}$ is in q_1 of egress port $[SW_1, ES_3]$ from t_0 until t_1 where transmission of $f_{1,1}^{[SW_1,ES_3]}$ is initiated.

Flow s_1, with period 100 µs, has three repetitions within the hyperperiod of 300 µs, while s_2, with period 150 µs, has two repetitions. Notice, how the start time of a frame is at the same offset in each period. For instance $f_{1,1}^{[ES_1,SW_1]}$ is transmitted at times 0 µs, 100 µs, and 200 µs, corresponding to $f_{1,1}^{[ES_1,SW_1]}.\phi =$ 0 µs. This restriction is the reason for the gap between $f_{2,2}^{[SW_1,ES_3]}$ and $f_{2,3}^{[SW_1,ES_3]}$ in the first repetition of s_2. In the second repetition of s_2 the gap is required to prevent conflicting with $f_{1,1}^{[SW_1,ES_3]}$. The model does not allow for different ϕ values in different repetitions.

There is an equivalence between the set of GCLs for all egress ports and a schedule like the one presented in Fig. 7. That is, a set of GCLs can be constructed from a schedule and vice versa. Table 7 illustrates the equivalence by presenting the GCL for egress port $[SW_1, ES_3]$. At time $t_0 = 0$ µs the gates of q_1 and q_2 for scheduled traffic are closed while the gates of all remaining queues are open to allow transmission of lower-priority traffic. At time t_1, q_1 is opened to transmit

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
q_1	0	1	0	0	0	0	1	0	0	0	1	0	0
q_2	0	0	1	0	1	0	0	0	1	0	0	1	0
remaining	1	0	0	1	0	1	0	1	0	1	0	0	1

Table 7: GCL for egress port $[SW_1, ES_3]$ for the schedule in Fig. 7. 1 and 0 represent open and closed gates, respectively.

$f_{1,1}^{[SW_1,ES_3]}$. At this time all other queues are closed to avoid interference from frames in other queues. At time t_2, $f_{1,1}^{[SW_1,ES_3]}$ has finished transmission and it is safe to open q_2 instead to initiate transmission of $f_{1,1}^{[SW_1,ES_3]}$ and $f_{2,2}^{[SW_1,ES_3]}$ after which q_2 is closed (at time t_3) and the remaining queues are opened for lower-priority traffic. The mutual exclusion of TT queues is what ensures determinism, and the remaining queues should be opened whenever the TT queues are closed to ensure a high QoS for other traffic types.

For the remainder of this report we will refer to schedules rather than tables of GCLs. Schedules are much better at visualizing the relationship between different links and egress ports.

12 Schedule Feasibility

A schedule must satisfy certain constraints in order to ensure feasibility. Some constraints enforce physical hardware limitations, while others provide real-time capabilities for time-critical traffic. In combination, they define the solution space of feasible schedules, i.e. schedules that are both realizable in physical TSN networks and meet the time-sensitive requirements. The following sections explain the constraints that define feasible schedules in this report.

12.1 Physical Properties

The solution space for feasible schedules is limited by some physical networking properties. They are listed below.

12.1.1 Link Congestion

A data link is limited by its hardware to only transmit a single frame at a time, i.e., frames on the same link cannot overlap in the time domain. This corresponds to the property that boxes on the same row of Fig. 7 do not overlap. The link can be seen as a critical section, that can only be occupied by a single frame at a time.

12.1.2 Flow Transmission

A switch cannot forward a frame until the entire frame has been buffered in the switch. This introduces a forwarding delay for each hop from source to destination. Due to the small synchronization error of the clocks between devices, the exact time when the entire frame has been received in a particular switch is unknown. Consequently, the time for forwarding the frame on the next link should take into account the worst-case synchronization error, δ. For instance, in Fig. 7, $f_{1,1}^{[ES_1,SW_1]}$ is transmitted on $[ES_1, SW_1]$ between $t_0 = 0\,\mu s$ and $t_1 = 12.336\,\mu s$. The same frame on the next link, $f_{1,1}^{[SW_1,ES_3]}$ does not initiate transmission until $t_1 = 18\,\mu s$, because it must wait for $f_{1,1}^{[ES_1,SW_1]}$ to be fully received in SW_1 even with maximum synchronization error. Without loss of generality, we assume $\delta = 5.008\,\mu s$ throughout the report, which means that transmission of $f_{1,1}^{[SW_1,ES_3]}$ must start after $12.336\,\mu s + 5.008\,\mu s = 17.344\,\mu s$.

12.1.3 FIFO Queuing

Frames are selected from egress port queues in a first in, first out (FIFO) fashion, i.e., frames in a queue should be forwarded in the same order they arrive in. There could be other physical properties related to the queues, for instance the maximum number of queued frames. However, for simplicity we assume that there is no limit on the queue size.

12.2 Time-Sensitive Requirements

The schedule must satisfy timeliness requirements for time-critical communication to achieve bounded latency and minimal jitter. The requirements are listed below.

12.2.1 Bounded Latency

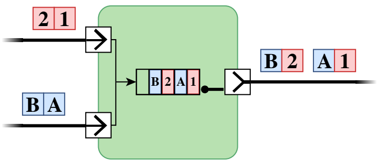
All time-critical messages must arrive within their relative deadline, i.e., the end-to-end latency from the time the transmission is initiated until it is fully received cannot exceed the deadline.

12.2.2 Minimal Jitter

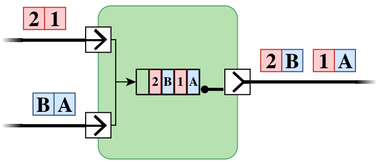
We define jitter as deviations in arrival time of messages from one period to another. A basic principle in the presented frame model is that each frame is transmitted at the same offset in every period. This effectively means that the model enforces minimal jitter, because deviations in arrival time can only be caused by mechanisms not directly captured by the model, such as the clock synchronization error.

12.2.3 Deterministic Queues

When a frame is scheduled to be transmitted on a link in a given time interval, the corresponding GCL is defined to open the associated gate in that interval. Suppose something goes wrong, so the frame is not fully received, or is not the first frame in the queue as expected. Then the link transmits the wrong frame or remains idle when it should be transmitting. Consequently, nondeterminism is introduced, which means timeliness is compromised.

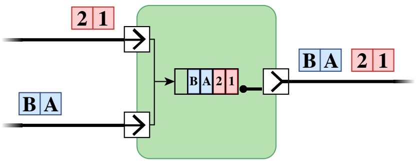


(a) Red flow arrives first.

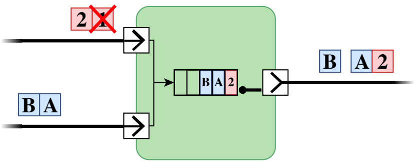


(b) Blue flow arrives first.

Figure 8: Nondeterministic queuing order because the scheduled arrival times vary with the clock synchronization error.



(a) No frame-loss.



(b) Frame “1” is lost.

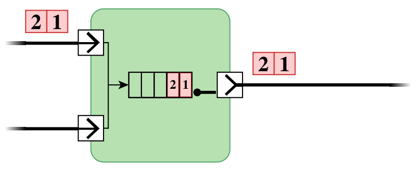
Figure 9: Nondeterminism caused by the loss of frame “1”. Jitter is introduced for the blue flow because its frames share queue with the lost frame.

Nondeterminism can occur in the queues in two scenarios:

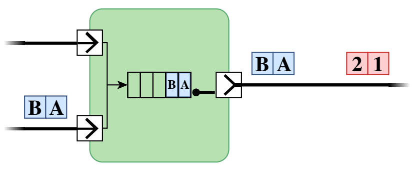
(i) Frames arriving on different ingress ports are scheduled to arrive at roughly the same time. Due to the clock synchronization error, the order in which frames are queued is nondeterministic, and could vary from period to period. This is illustrated in Fig. 8.

(ii) If a frame is lost for some reason, it of course introduces nondeterminism for that particular flow, but it could also affect frames of other flows if they are in the same queue. Fig. 9 illustrates the situation, where the loss of frame of one flow affects the transmission time of another flow.

To enforce determinism, queue-sharing flows should be scheduled carefully. Frames of such flows must be scheduled so their arrival times are so far apart that the arrival order is deterministic even with the maximal clock synchronization error, and such that only frames of one of the flows are present in the queue at a time. Analogously to the *link congestion* property, a queue can be considered a critical section, which can only be occupied by frames from a single flow at a time. Fig. 10 illustrates the solution.



(a) Red flow occupies queue.



(b) Blue flow arrives when queue is empty.

Figure 10: Determinism is enforced by scheduling frames of queue-sharing flows far apart. The arrival time of the blue flow is delayed until the red flow has left the queue.

13 Problem Formulation

In this report, we address the problem of scheduling Time-Triggered (TT) flows on a network architecture. The problem is formulated as follows: Given a TSN network topology $\mathcal{G}(\mathbf{E}, \mathbf{V})$, and a set of TT flows $\mathcal{S}$, determine Gate-Control Lists (GCLs) for each egress port in the network.

As established in Sect. 10, this corresponds to finding a feasible schedule, i.e., feasible assignments for the following two sets of variables:

(i) A queue $s_i^{[v_a,v_b]}.\rho$ in egress port $[v_a, v_b]$ for each flow instance $s_i^{[v_a,v_b]} \in \mathcal{I}$.

(ii) An offset $f_{i,m}^{[v_a,v_b]}.\phi$ for the periodic transmission on data link $[v_a, v_b]$ for each frame $f_{i,m}^{[v_a,v_b]} \in \mathcal{F}$.

A feasible schedule satisfies all hardware-imposed constraints while meeting safety-critical timeliness requirements as described in Sect. 12.

We wish to determine a solution such that two objectives are minimized: (1) The number of queues used by TT flows, and (2) the end-to-end latency for each flow. The reason, for minimizing the number of TT queues, is to maximize the number of queues that remain available for lower-priority traffic, such as AVB and BE. End-to-end latency should be minimized because it is an important Quality of Service (QoS) metric in most industrial applications. Furthermore, a minimized end-to-end latency means frames spend less time waiting in queues, which help reduce memory requirements for queues.

14 Flow Routing

To simplify the problem, we assume that route $s_i.r$ of each flow s_i is given. The routes can be preprocessed using Dijkstra's algorithm [35] to determining a shortest path in the weighted network graph $\mathcal{G}(\mathbf{E}, \mathbf{V})$. The weight of edge $[v_a, v_b]$ is computed based on the transmission rate $[v_a, v_b].s$ and propagation delay $[v_a, v_b].d$, such that a shortest path represents the fastest way to transmit messages from sender to receiver, thereby enabling low end-to-end latencies.

This preprocessing step reduces problem complexity, but could result in sub-optimal routing in network topologies with redundant routes. However, in this report, we focus on topologies with only a single route connecting any pair of end systems.

15 Motivational Example

The running example from Sect. 11 may be used as a motivational example. We have $\mathcal{S} = \{s_1, s_2\}$, and the schedule in Fig. 7 represents a feasible solution to the presented problem. The solution is given by the assignments in Eq. 10–14:

$$s_1^{[SW_1,ES_3]}.\rho = 1, \qquad s_2^{[SW_1,ES_3]}.\rho = 2 \tag{10}$$
$$f_{1,1}^{ES_1,SW_1}.\phi = 0\,\mu\text{s}, \qquad f_{1,1}^{SW_1,ES_3}.\phi = 18\,\mu\text{s} \tag{11}$$
$$f_{2,1}^{ES_2,SW_1}.\phi = 13\,\mu\text{s}, \qquad f_{2,1}^{SW_1,ES_3}.\phi = 31\,\mu\text{s} \tag{12}$$
$$f_{2,2}^{ES_2,SW_1}.\phi = 26\,\mu\text{s}, \qquad f_{2,2}^{SW_1,ES_3}.\phi = 44\,\mu\text{s} \tag{13}$$
$$f_{2,3}^{ES_2,SW_1}.\phi = 63\,\mu\text{s}, \qquad f_{2,3}^{SW_1,ES_3}.\phi = 81\,\mu\text{s} \tag{14}$$

Queue assignments for end systems are not shown, as they are trivially $\rho = 1$ because end systems are modeled with only one queue. Every egress port transmitting TT traffic requires at least one queue for storing TT frames. Hence, the interesting metric is the total number of *excess* queues used for TT traffic. We denote this metric K. The motivational example uses one excess queue, namely q_2 in egress port $[SW_1, ES_3]$. Hence, K = 1.

The end-to-end latency for a flow is defined as the time from transmission starts in the sending end system, and until the last frame has been fully received in the receiving end system. Every frame in the example has a duration of 12.336 µs, thus, the end-to-end latencies for s_1 and s_2, denoted λ_1 and λ_2, respectively, are calculated in Eq. 15 and 16.

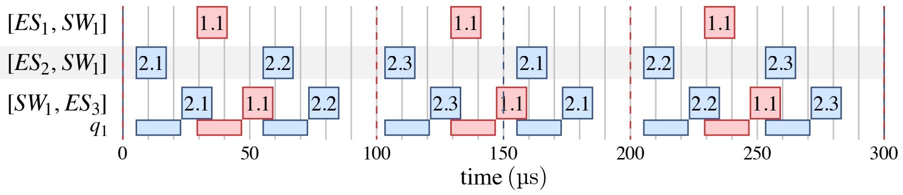
$$\begin{aligned}\lambda_1 &= (f_{1,1}^{SW_1,ES_3}.\phi + f_{1,1}^{SW_1,ES_3}.L) - f_{1,1}^{ES_1,SW_1}.\phi \qquad (15)\\ &= (18\,\mu s + 12.336\,\mu s) - 0\,\mu s = 30.336\,\mu s\\ \lambda_2 &= (f_{2,3}^{SW_1,ES_3}.\phi + f_{1,1}^{SW_1,ES_3}.L) - f_{2,1}^{ES_2,SW_1}.\phi \qquad (16)\\ &= (81\,\mu s + 12.336\,\mu s) - 13\,\mu s = 80.336\,\mu s\end{aligned}$$

The *link congestion* (Sect. 12.1.1) and *flow transmission* (Sect. 12.1.2) constraints impose a lower bound on the end-to-end latency. The lower bound is caused by the fact that each frame must be fully received in a switch before it can be forwarded, and the fact that links cannot transmit multiple frames at once. The lower bound depends on the route, and the number and duration of the frames in a flow. We denote the lower bound for flow s_i with $\underline{\lambda}_i$. The lower bounds for the two flows are $\underline{\lambda}_1 = 30.336\,\mu s$ and $\underline{\lambda}_2 = 56.336\,\mu s$. Sect. 18.1 explains how the bounds are calculated. Similar to queue metric, the interesting metric is the total amount of additional latency compared to the lower bounds. The accumulated additional latency, Λ, is calculated in Eq. 17.

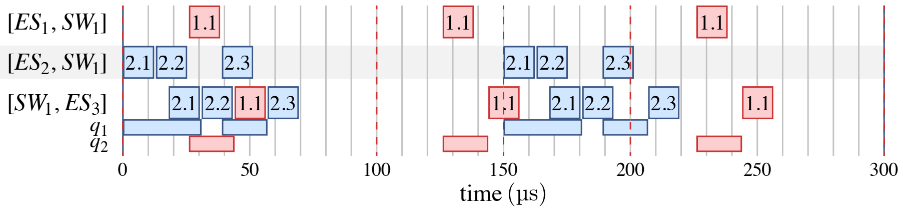
$$\begin{aligned}\Lambda &= (\lambda_1 - \underline{\lambda}_1) + (\lambda_2 - \underline{\lambda}_2) \qquad (17)\\ &= (30.336\,\mu s - 30.336\,\mu s) + (80.336\,\mu s - 56.336\,\mu s) = 24\,\mu s\end{aligned}$$

The quality of a solution can be described by the tuple (K, Λ). For the solution in Eq. 10 - Eq. 14 the objective value is $(\mathrm{K} = 1, \Lambda = 24\,\mu s)$. In general, many feasible solutions exist for the same problem instance. Fig. 16 shows two different feasible solutions. One improves K, the other improves Λ. In fact, Fig. 11a is optimal with respect to K and Fig. 11b is optimal with respect to Λ. Even for this small example it is not immediately apparent that the end-to-end latency of Fig. 11b cannot be improved. At first sight, it seems possible to eliminate the gap between $f_{2,2}^{[SW_1,ES_3]}$ and $f_{2,3}^{[SW_1,ES_3]}$, thereby reducing Λ to zero. However, it is not possible to schedule s_1 such that it does not interfere with the three frames of s_2.

This motivational example shows that it is difficult to find high-quality solutions for the presented scheduling problem — even for very small problem instances.



(a) Optimal with respect to number of excess queues. Objective value: $(0, 72\,\mu s)$.



(b) Optimal with respect to end-to-end latency. Objective value: $(1, 13\,\mu s)$.

Figure 11: Different feasible schedules for the same problem instance. (a) is better than Fig. 7 in terms of number of excess queues, (b) is better in terms of end-to-end latency.

16 Problem Complexity

The presented TT scheduling problem is a multi-objective combinatorial optimization problem. In the following we proof its NP-completeness.

There is a polynomial number of constraints defining the solution space of feasible schedules. For instance, the *link congestion* property requires one constraint for each pair of link-sharing frames. Each constraint can be checked in constant time, hence it can be verified in polynomial time if a schedule is feasible. This shows that the TT scheduling problem is in NP.

Scheduling frames within their deadlines is similar to the flow-shop scheduling problem which is known to be NP-complete [36]. In fact, the decision problem of flow-shop scheduling reduces to the problem of finding a feasible TT schedule. In the flow-shop scheduling problem, m jobs are scheduled on n machines. Each job consists of n operations, one for each machine. The operations must be carried out in order, such that the first operation of a job is performed on the first machine, the second operation on the second machine, etc. A feasible solution must satisfy two constraints:

(i) Machines perform at most one operation at a time.

(ii) One operation of a job must finish before the next operation can start execution on the next machine.

The execution time of each operation of each job is given. The decision problem is to determined an arrangement of operations, such that the makespan, i.e., total job execution time, is less than some constant D.

The TT scheduling problem is directly mapped to the flow-shop scheduling problem. Links correspond to machines, flows to jobs, and frames to operations. Hence, instance X for the flow-shop scheduling problem is transformed into instance Y of the TT scheduling problem with m flows and n links. Flows consist of a single frame, and they all have the same route going through the n links. The period and deadline of all flows are set to D, and all flows are assigned unique queues. Constraint (i) corresponds to the *link congestion* property, and constraint (ii) corresponds to the *flow transmission* property.

With this mapping, X has a solution with makespan less than D if and only if the frames in Y are schedulable within their deadline D. Consequently, the flow-shop scheduling problem reduces to the TT scheduling problem in polynomial time which concludes the proof that the TT scheduling problem is NP-complete. In additional to finding a feasible schedule, we are interested in finding schedules that minimize queue usage and end-to-end latencies. This only adds to the complexity of the problem.

Assuming that P $\neq$ NP, the scheduling problem cannot be solved to optimality with any polynomial-time algorithm. The number of solutions increases exponentially with the input size. Since most industrial applications require much larger schedules than the ones in Sect. 15, advanced optimization strategies are required to solve the problem.

In the following three sections we present different approaches to solving the TT scheduling problem. All three represent different compromises between scalability and solution quality. The first, an ILP approach, has a worst-case exponential running time but with proven optimality. The second, a heuristic approach, has a polynomial running time but gives no guarantees regarding solution quality. The third, a metaheuristic approach, fills the gap in between the two other approaches. It iteratively improves solution quality starting from an initial solution. By using a fast algorithm for generating the initial solution — such as the heuristic approach — it remains scalable while yielding high quality solutions due to the iterate improvement.

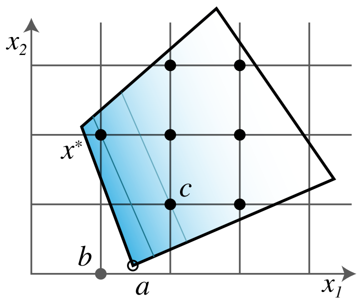


Figure 12: Two-dimensional polytope for ILP minimization problem.

17 ILP Strategy

Integer Linear Programming (ILP) is a mathematical optimization program consisting of integer variables, a linear objective function, and linear inequalities. The linear inequalities make up a polyhedron defining the complete solution space, X, for the problem instance. A feasible solution, $x \in X$, is an assignment of values to integer variables, such that all inequalities are satisfied. For minimization problems, an *optimal* solution, x^*, is a feasible solution with minimum objective value, and conversely for maximization problems.

The objective function, z, is a function mapping solutions to real numbers, i.e., $z : x \to \mathbb{R}$. For minimization problems, z^* denotes the minimum objective value, hence $z^* = z(x^*) = \min\{z(x) : x \in X \}$. Solving an ILP problem boils down to determining z^* and x^*.

Fig. 12 illustrates the principle of a bounded polyhedron (polytope) defining the solution space of two integer variables x_1 and x_2. Only the marked integer solutions inside the polytope are feasible. The gradient visualizes the objective values for different locations inside the polytope. The darker the color, the smaller the objective value, hence, the corresponding minimization problem is to determine an integer solution x^* at the "darkest" location. Here $x^* = (1, 2)$.

Intuitively, it seems like a reasonable solution strategy to find the minimum *real*-valued solution inside the polytope, which can be achieving in polynomial time, and round to nearest integer solution. However, as shown in Fig. 12 this is a flawed strategy. Point a depicts the minimum real-valued solution, the optimal solution of the so-called *linear relaxation.* Rounding a to nearest integer solution yields $b = (1, 0)$, which is not inside the polytope, thus infeasible. Rounding to a solution inside the polytope, such as $c = (2, 1)$, does not guarantee optimality as the figure shows.

Many NP-complete problems can be formulated and solved as ILP problems, and as such ILP itself is NP-hard. Commercial state-of-the-art solvers like CPLEX [37] and Gurobi [38] are designed to solve complicated ILP problems with thousands of variables and constraints. The solvers utilize many different techniques (e.g., branch and bound) for limiting the search space exploration, thereby improving performance. However, this does not change the fact that — in the worst case — solving an ILP problem requires full enumeration of all feasible solutions. The total number of feasible solutions increases exponentially with the input size for NP-complete problems, including the TT scheduling problem.

18 Objective

As motived in Sect. 15, the objective of the TT scheduling problem is to minimize the metric pair (K, Λ), where K denotes the total number of *excess* queues used for TT flows, and Λ denotes the total *extra* end-to-end latency introduced by interfering TT flows. The calculation of K and Λ are shown in Eq. 18 and 19.

$$\mathrm{K} = \sum_{[v_a, v_b] \in \mathbf{E}} (\kappa_{a,b} - 1) \tag{18}$$

$$\Lambda = \sum_{s_i \in \mathcal{S}} (\lambda_i - \underline{\lambda}_i) \tag{19}$$

The variable $\kappa_{a,b} \in \mathbb{N}$ denotes the number of queues used by TT flows in egress port $[v_a, v_b]$. Every TT-enable egress port has at least one TT queue, and as such the term $(\kappa_{a,b} - 1)$ accounts for the number of excess queues in $[v_a, v_b]$. The variable $\lambda_i \in \mathbb{N}$ denotes the end-to-end latency for flow s_i, and $\underline{\lambda}_i$ denotes a lower bound on the latency, that is, the latency of flow s_i, as if no other flows were interfering. Thus, $(\lambda_i - \underline{\lambda}_i)$ evaluates to the extra end-to-end latency for s_i.

Both metrics, K and Λ, should be as low as possible, which makes the problem a multi-objective minimization problem. ILP optimizes a single objective value, hence the two metrics are combined into one objective function by introducing weights c_1 and c_2 for K and Λ, respectively. The objective value is defined in Eq. 20.

$$\begin{aligned} z &= c_1 \cdot \mathrm{K} + c_2 \cdot \Lambda \\ &= c_1 \sum_{[v_a,v_b]\in\mathbf{E}} (\kappa_{a,b} - 1) + c_2 \sum_{s_i\in\mathcal{S}} (\lambda_i - \underline{\lambda}_i) \end{aligned} \tag{20}$$

For instance, by setting c_1 sufficiently large, and $c_2 = 1$, the model is configured to prioritize queue minimization over end-to-end latency, i.e., first the number of queues is optimized, then end-to-end latency. This will yield the schedule with minimal end-to-end latency among all schedules using a minimum number of queues.

18.1 Lower Bound on End-To-End Latency

A lower bound, $\underline{\lambda}_i$, for the end-to-end latency of flow s_i is calculated using a dynamic programming approach. Recall that s_i is sent on route, $s_i.r$, from sending link $s_i.s$ to receiving link $s_i.t$, and consists of $s_i.k$ frames. We introduce the notation $f_{i,m}^{[v_a,v_b]}.\underline{\phi}$ to denote a lower bound on the offset of frame $f_{i,m}^{[v_a,v_b]}$, i.e., a lower bound on $f_{i,m}^{[v_a,v_b]}.\phi$. The lower bound is the earliest possible offset of $f_{i,m}^{[v_a,v_b]}$ if no other flows interfere. Hence, for the first frame on the sending link we have:

$$f_{i,1}^{s_i.s}.\underline{\phi} = 0 \tag{21}$$

Due to the *link congestion* property (Sect. 12.1.1), the offsets of the remaining frames on the sending link is lower bounded by the completion time of the previous frame. This is formulated in Eq. 22.

$$f_{i,m}^{s_i.s}.\underline{\phi} = f_{i,m-1}^{s_i.s}.\underline{\phi} + \lceil f_{i,m-1}^{s_i.s}.L \rceil \quad \forall m \in \{2, 3, \ldots, s_i.k\} \tag{22}$$

The duration of frame $f_{i,m-1}^{s_i.s}$ is rounded up because $f_{i,m}^{s_i.s}.\phi$ only takes integral values.

Due to the *flow transmission* property (Sect. 12.1.2), the offset of the first frame of link $[v_a, v_b]$ is lower bounded by the completion time (plus maximum synchronization error) of the same frame on the previous link $[v_x, v_a]$. This is formulated in Eq. 23.

$$f_{i,1}^{[v_a,v_b]}.\underline{\phi} = f_{i,1}^{[v_x,v_a]}.\underline{\phi} + \left\lceil f_{i,1}^{[v_x,v_a]}.L + \delta \right\rceil \quad \forall [v_a, v_b] \in s_i.r \setminus \{s_i.s\} \tag{23}$$

Eq. 21, 22, and 23 establish the base cases for the dynamic programming strategy. The offset for any other frame is lower bounded by both the completion time of the previous frame on the same link *and* the completion time (plus synchronization error) of the same frame on the previous link. The recursion is formulated in Eq. 24.

$$f_{i,m}^{[v_a,v_b]}.\underline{\phi} = \max\left\{ f_{i,m-1}^{[v_a,v_b]}.\underline{\phi} + \left\lceil f_{i,m-1}^{[v_a,v_b]}.L \right\rceil, f_{i,m}^{[v_x,v_a]}.\underline{\phi} + \left\lceil f_{i,m}^{[v_x,v_a]}.L + \delta \right\rceil \right\} \tag{24}$$

With the recursive definition of $f_{i,m}^{[v_a,v_b]}.\underline{\phi}$, the lower bound on the end-to-end latency is formulated as the difference between the completion time of the last frame on the last link, and the offset of the first frame on the first link, as shown in Eq. 25.

$$\underline{\lambda}_i = f_{i,k}^{s_i.t}.\underline{\phi} + f_{i,k}^{s_i.t}.L - f_{i,1}^{s_i.s}.\underline{\phi} = f_{i,k}^{s_i.t}.\underline{\phi} + f_{i,k}^{s_i.t}.L \tag{25}$$

19 ILP Model

The two primary sets of decision variables of the ILP model are the frame offsets and the flow instance queue assignments. They are both integer variables, hence $f_{i,m}^{[v_a,v_b]}.\phi \in \mathbb{N}$ for all $f_{i,m}^{[v_a,v_b]} \in \mathcal{F}$ and $s_i^{[v_a,v_b]}.\rho \in \mathbb{N}$ for all $s_i^{[v_a,v_b]} \in \mathcal{I}$.

The ILP model is made up of three parts. The first part is formulated in Eq. 26 – 31, with Eq. 26 being the objective function derived in Sect. 18.

$$\min \quad c_1 \sum_{[v_a,v_b]\in\mathbf{E}} (\kappa_{a,b} - 1) + c_2 \sum_{s_i\in\mathcal{S}} (\lambda_i - \underline{\lambda}_i) \tag{26}$$

$$\text{s.t.} \quad \kappa_{a,b} \geq s_i^{[v_a,v_b]}.\rho \qquad \forall s_i^{[v_a,v_b]} \in \mathcal{I} \tag{27}$$

$$\lambda_i = f_{i,k}^{s_i.t}.\phi + f_{i,k}^{s_i.t}.L - f_{i,1}^{s_i.s}.\phi \qquad \forall s_i \in \mathcal{S} \tag{28}$$

$$\lambda_i \leq s_i.D \qquad \forall s_i \in \mathcal{S} \tag{29}$$

$$f_{i,m}^{[v_a,v_b]}.\phi \leq s_i.T - f_{i,m}^{[v_a,v_b]}.L \qquad \forall f_{i,m}^{[v_a,v_b]} \in \mathcal{F} \tag{30}$$

$$f_{i,m}^{[v_a,v_b]}.\phi \geq f_{i,m}^{[v_x,v_a]}.\phi + f_{i,m}^{[v_x,v_a]}.L + \delta \qquad \forall f_{i,m}^{[v_a,v_b]}, f_{i,m}^{[v_x,v_a]} \in \mathcal{F}^2 \tag{31}$$

The semantics of variables $\kappa_{a,b}$ and λ_i, as described in Sect. 18, are implemented by constraints Eq. 27 and 28, respectively. Eq. 27 ensures that each $\kappa_{a,b}$ is lower bounded by every queue ID assigned to a flow instance on $[v_a, v_b]$. Eq. 28 defines λ_i to be the time between transmission of the first frame, on the first link, until the last frame is fully received on the last link of s_i, analogously to Eq. 25.

Eq. 29 models the *bounded latency* requirement (Sect. 12.2.1), i.e., it enforces that every flow arrives within its deadline. Eq. 30 defines an upper bound on frame offsets, namely, that every frame has to be fully scheduled within its period as defined in Sect. 10.2. Eq. 31 captures the *flow transmission* property (Sect. 12.1.2). That is, frame $f_{i,m}^{[v_a,v_b]}$ on link $[v_a, v_b]$ must start *after* the same frame, $f_{i,m}^{[v_x,v_a]}$, on the previous link, $[v_x, v_a]$, has been fully received in switch v_a, even when considering the worst-case synchronization error, δ.

The two remaining parts are slightly more involved and are concerned with capturing the *link congestion* property (Sect. 12.1.1), and the *deterministic queues* requirement (Sect. 12.2.3). These are described in the following.

19.1 Link Congestion

The *link congestion* property expresses that each link can transmit at most one frame at a time, i.e., frames on the same link cannot overlap in the time domain. Eq. 32 models this by enforcing an in-order transmission of frames from the same flow.

$$f_{i,m}^{[v_a,v_b]}.\phi + f_{i,m}^{[v_a,v_b]}.L \leq f_{j,n}^{[v_a,v_b]}.\phi \quad \forall f_{i,m}^{[v_a,v_b]}, f_{i,n}^{[v_a,v_b]} \in \mathcal{F}^2, m < n \tag{32}$$

It models that frames $f_{i,1}^{[v_a,v_b]}$, $f_{i,2}^{[v_a,v_b]}$, ..., $f_{i,m}^{[v_a,v_b]}$, ..., $f_{i,k}^{[v_a,v_b]}$ are transmitted ordered by their index, and frame m cannot initiate transmission until after frame $m-1$ is fully transmitted.

Frames of two different flows, s_i and s_j, might have different periods, i.e., $s_i.T \neq s_j.T$. Consequently, they are repeated a different number of times within their hyperperiod. We denote the hyperperiod of the two flows by $\text{hp}_{i,j} = \text{lcm}(s_i.T, s_j.T)$, where $\text{lcm}(s_i.T, s_j.T)$ denotes the Least Common Multiple (LCM) of the two periods. The set of translations of frames of flow s_i and s_j within the hyperperiod are denoted with A and B, respectively, as formulated in Eq. 33.

$$A = \left\{0, 1, \ldots, \tfrac{\mathrm{hp}_{i,j}}{s_i.T} - 1\right\}, \quad B = \left\{0, 1, \ldots, \tfrac{\mathrm{hp}_{i,j}}{s_j.T} - 1\right\} \tag{33}$$

Every frame $f_{i,m}^{[v_a,v_b]}$ of flow s_i, and every $f_{j,n}^{[v_a,v_b]}$ of s_j, are repeatedly transmitted within the hyperperiod at times:

$$\alpha \cdot s_i.T + f_{i,m}^{[v_a,v_b]}.\phi \quad \forall \alpha \in A, \quad \text{and} \quad \beta \cdot s_j.T + f_{j,n}^{[v_a,v_b]}.\phi \quad \forall \beta \in B$$

The frames could potentially overlap in the time domain for any choice of α and β. Constraints Eq. 34 and 35 prevent such overlaps.

$$\alpha \cdot s_i.T + f_{i,m}^{[v_a,v_b]}.\phi + f_{i,m}^{[v_a,v_b]}.L \leq \beta \cdot s_j.T + f_{j,n}^{[v_a,v_b]}.\phi + M \cdot \sigma \tag{34}$$

$$\beta \cdot s_j.T + f_{j,n}^{[v_a,v_b]}.\phi + f_{j,n}^{[v_a,v_b]}.L \leq \alpha \cdot s_i.T + f_{i,m}^{[v_a,v_b]}.\phi + M\,(1-\sigma) \tag{35}$$

$$\forall f_{i,m}^{[v_a,v_b]}, f_{j,n}^{[v_a,v_b]} \in \mathcal{F}^2, \forall \alpha \in A, \forall \beta \in B$$

Eq. 34 expresses that $f_{i,m}^{[v_a,v_b]}$ must finish before $f_{j,n}^{[v_a,v_b]}$ starts for each choice of α and β, and Eq. 35 expresses the reverse situation. An auxiliary binary variable $\sigma \in \{0,1\}$ is introduced to model disjunction, such that only one of the two constraints has to be satisfied. M represents a (theoretically) infinitely large constant, which causes either the inequality of Eq. 34 or Eq. 34 to be trivially satisfied if $\sigma = 1$ or $\sigma = 0$, respectively. Note that there is such a σ-variable for each choice of $f_{i,m}^{[v_a,v_b]}$, $f_{j,n}^{[v_a,v_b]}$, α, and β, but this is left out to simplify notation.

19.2 Deterministic Queues

As concluded in Sect. 12.2.3, deterministic queues are achieved by allowing at most one flow present in a queue at any given time. Suppose two flows, s_i and s_j, share the same queue in switch v_a. Determinism is enforced by restricting all frames of s_j from entering the queue in v_a while a frame of s_j is queued, and vice versa. Eq. 36 and 37 model the restriction in the simplest case where s_i and s_j arrive on the same ingress port, $[v_x, v_a]$.

$$\begin{aligned}&\alpha \cdot s_i.T + f_{i,m}^{[v_a,v_b]}.\phi \leq \\ &\qquad \beta \cdot s_j.T + f_{j,n}^{[v_x,v_a]}.\phi + M\left(\omega + \epsilon_{i,j}^{[v_a,v_b]} + \epsilon_{j,i}^{[v_a,v_b]}\right)\end{aligned} \tag{36}$$

$$\begin{aligned}&\beta \cdot s_j.T + f_{j,n}^{[v_a,v_b]}.\phi \leq \\ &\quad \alpha \cdot s_i.T + f_{i,m}^{[v_x,v_a]}.\phi + M\left(1 - \omega + \epsilon_{i,j}^{[v_a,v_b]} + \epsilon_{j,i}^{[v_a,v_b]}\right)\end{aligned} \tag{37}$$

$$\forall f_{i,m}^{[v_x,v_a]}, f_{i,m}^{[v_a,v_b]}, f_{j,n}^{[v_x,v_a]}, f_{j,n}^{[v_a,v_b]} \in \mathcal{F}^4, \forall \alpha \in A, \forall \beta \in B$$

Eq. 36 expresses the case where $f_{i,m}^{[v_a,v_b]}$ is scheduled such that it leaves the queue in v_a before $f_{j,n}^{[v_x,v_a]}$ enters, and vice versa in Eq. 37. The auxiliary binary variable $\omega \in \{0,1\}$, and the large constant M, are used to implement disjunction in the same way as in Sect. 19.1.

Another auxiliary binary variable $\epsilon_{i,j}^{[v_a,v_b]}$ [39] is introduced to capture whether or not $s_i^{[v_a,v_b]}$ and $s_j^{[v_a,v_b]}$ are assigned the same queue. The variable is 1 if and only if $s_i^{[v_a,v_b]}$ is assigned a queue with ID strictly less than the queue ID of $s_j^{[v_a,v_b]}$. The formal definition is presented in Eq. 38.

$$\epsilon_{i,j}^{[v_a,v_b]} = \begin{cases} 1, & s_i^{[v_a,v_b]}.\rho < s_j^{[v_a,v_b]}.\rho \\ 0, & \text{otherwise} \end{cases} \tag{38}$$

As such, the sum $\epsilon_{i,j}^{[v_a,v_b]} + \epsilon_{j,i}^{[v_a,v_b]}$ equals 1 if and only if s_i and s_j are assigned *different* queues in egress port $[v_a, v_b]$. In this case both constraints are trivially satisfied by addition of the huge constant M. The constraints required to implement the described semantics for ϵ are formulated in Eq. 39 and 40.

$$s_j^{[v_a,v_b]}.\rho - s_i^{[v_a,v_b]}.\rho - M\left(\epsilon_{i,j}^{[v_a,v_b]} - 1\right) \geq 1 \tag{39}$$

$$s_j^{[v_a,v_b]}.\rho - s_i^{[v_a,v_b]}.\rho - M \cdot \epsilon_{i,j}^{[v_a,v_b]} \leq 0 \tag{40}$$

$$\forall s_i^{[v_a,v_b]}, s_j^{[v_a,v_b]} \in \mathcal{I}^2$$

In the case that s_i and s_j arrive on different ingress ports, the constraints need to take into account the worst-case synchronization error. Suppose that s_i arrives on ingress port $[v_x, v_a]$ and s_j arrives on $[v_y, v_a]$. Then the internal clocks of v_x and v_y could differ by as much as δ, causing frames to be forwarded later or earlier than expected. Consequently, the arrival times of frames at the ingress ports is further restricted by the size of δ. This is modeled in constraints Eq. 41 and 42.

$$\alpha \cdot s_i.T + f_{i,m}^{[v_a,v_b]}.\phi + \delta \leq$$

$$\beta \cdot s_j.T + f_{j,n}^{[v_y,v_a]}.\phi + M\left(\omega + \epsilon_{i,j}^{[v_a,v_b]} + \epsilon_{j,i}^{[v_a,v_b]}\right) \tag{41}$$

$$\beta \cdot s_j.T + f_{j,n}^{[v_a,v_b]}.\phi + \delta \leq$$

$$\alpha \cdot s_i.T + f_{i,m}^{[v_x,v_a]}.\phi + M\left(1 - \omega + \epsilon_{i,j}^{[v_a,v_b]} + \epsilon_{j,i}^{[v_a,v_b]}\right) \tag{42}$$

$$\forall f_{i,m}^{[v_x,v_a]}, f_{i,m}^{[v_a,v_b]}, f_{j,n}^{[v_y,v_a]}, f_{j,n}^{[v_a,v_b]} \in \mathcal{F}^4, \forall \alpha \in A, \forall \beta \in B$$

The *FIFO queuing* property (Sect. 12.1.3) is implicitly enforced by constraints Eq. 32, 36, 37, 41, and 42. The first constraint defines a static ordering of frames within the same flow, resulting in FIFO queuing for such frames. The remaining constraints restrict that only one flow can occupy a queue at a time, preventing frames of other flows from interfering with the queueing order. Thus, an explicit FIFO queuing constraint is redundant. In combination, the entire set of presented constraints define the polytope for feasible solutions, over which the objective function is minimized. The presented ILP model is a refinement of the model we present in [27]. The constraint definitions are inspired by the SMT-model in [26].

20 Analysis

The polyhedron defining the feasible solutions consists of $O(|\mathcal{F}|^2)$ constraints, but the number of constraints is not directly related to the running time for solving the ILP model. Instead, the running time increases exponentially with the input size in the worst case. Hence, this method is well-suited for finding proven optimal solutions for small instances, but is not expected to be tractable for medium or large problem instances.

21 Implementation Details

The ILP model is implemented in Python using the PuLP package to model objective function, constraints, and decision variables. PuLP support a number of commercial and open-source solvers to optimize the modeled problem. We have primarily used the commercial solver Gurobi [38]. A time limit is given as an input to the solver. If the problem has not been solved to optimality within the time limit, the search terminates, and the current best feasible solution is returned along with a percentage gap to the lower bound. If nothing else is stated the time limit is set to four hours. Note that it could be the case that no feasible solution has been found within the time limit. In this case the ILP strategy fails to solve the problem, either because no solution exists, or because it was simply not given enough time.

22 Heuristic Strategy

We propose a heuristic strategy as a tractable alternative to the ILP approach presented in Sect. 17. The greedy scheduling algorithm presented in this section starts with an empty schedule, and iteratively expands the schedule with one flow at a time. Each flow is scheduled such that the number of queues and end-to-end latency is minimized.

The algorithm terminates when all flows have been scheduled, or when the current iteration fails to schedule a flow within its deadline. The advantage of this approach is that a very limited search space is explored in every iteration, and there is a finite number of iterations. Consequently, it scales well to large instances with many flows. The disadvantage is that each iteration does not consider the flows which are still to be scheduled, and therefore may make poor choices leading to suboptimal solutions, or it might fail solve some instances.

Algorithm 1 shows the overall strategy of the heuristic. As an input it takes the set of TT flows $\mathcal{S}$. The algorithm outputs a feasible solution. In the case of failure, it outputs a partial feasible schedule, consisting of the successfully scheduled flows until the point of failure.

The first step of the algorithm is to determine the order in which routes are to be scheduled (line 3). Flows are scheduled in the order of their deadlines, starting with the earliest deadline. Flows with early deadlines should be scheduled first to have the best chances of success. As a secondary sorting criteria, flows are sorted ascending by their period. A flow with low period is repeated more often within the hyperperiod, and thus, is more difficult to schedule. On the contrary, a high period flow has more flexibility to "fit around" lower period frames. To break ties, if the deadline and period of two flows are identical, we use the length of the routes, under the assumption that longer data paths are more difficult to schedule.

With the order in place, flows are considered one-by-one. The heuristic assumes that the primary objective is to minimize the number of queues used for TT flows, and the secondary objective is to minimize end-to-end latency. Therefore, it initially attempts to schedule the current flow using only the first queue for all flow instances (line 6). The subroutine SCHEDULEFLOW(s_i, x) on line 8 schedules s_i when considering the already scheduled flows of the partial solution

Algorithm 1 Greedy heuristic with minimal number of queues.

```
function GreedyHeuristic($\mathcal{S}$)
    $x \leftarrow \varnothing$
    $\mathcal{S}' \leftarrow$ SortFlows($\mathcal{S}$)
    for $s_i \in \mathcal{S}'$ do
        $success \leftarrow$ false
        $s_i^{[v_a,v_b]}.\rho \leftarrow 1 \quad \forall [v_a, v_b] \in s_i.r$
        repeat
            if ScheduleFlow($s_i, x$) = true then
                $x \leftarrow x \cup \{s_i\}$
                $success \leftarrow$ true
            else
                $[v_x, v_y] \leftarrow$ ConstrainingEgressPort($s_i, x$)
                $s_i^{[v_x,v_y]}.\rho \leftarrow s_i^{[v_x,v_y]}.\rho + 1$
                if $s_i^{[v_x,v_y]}.\rho > [v_x, v_y].c$ then                ▷ Failure
                    return $x$
        until $success$ = true
    return $x$
```

x. In case of success it returns **true**, otherwise it returns **false**. The internal workings of this subroutine are elaborated in Sect. 23.

Lines 7–16 contain a feedback loop which increments the queue ID of a relevant queue every time the algorithm fails to schedule s_i with the current queue assignments. The feedback loop repeats until a successful queue assignment has been found, or until running out of queues. If the scheduling succeeds, s_i is added to the partial solution x (line 9), if not, the algorithm returns the partial solution found so far (line 15).

23 Scheduling a Single Flow

Given queue assignments $s_i^{[v_a,v_b]}.\rho$ for each flow instance, and a partial solution x of previously scheduled flows, flow s_i should be scheduled such that the schedule remains feasible, and the end-to-end latency is minimized. We start out by explaining a method for determining the feasible regions for the offsets of s_i, then we present an as-soon-as-possible (ASAP) strategy for scheduling the frames of

s_i, and finally we discuss different ways of minimizing end-to-end latency as a post-processing step.

23.1 Feasible Regions

The feasible region for $f_{i,m}^{[v_a,v_b]}.\phi$ is a set of intervals where $f_{i,m}^{[v_a,v_b]}$ can be scheduled without violating the feasibility of the existing partial schedule, x. For any point in time t in the feasible region for $f_{i,m}^{[v_a,v_b]}.\phi$, the following three conditions hold:

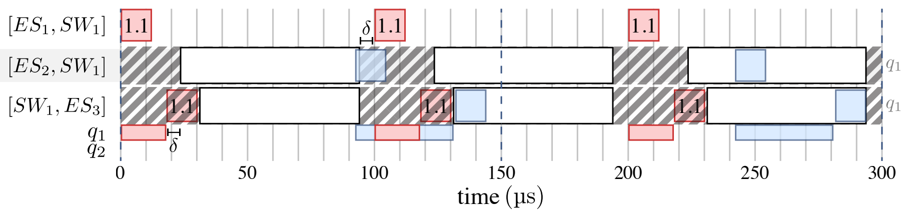
(i) The link $[v_a, v_b]$ is available in the interval $[t; t + f_{i,m}^{[v_a,v_b]}.L]$, i.e., no frames in x are transmitted on the link in this interval.

(ii) The assigned queue in egress port $[v_a, v_b]$ is available at time t. If other frames, arriving from different devices than $f_{i,m}^{[v_a,v_b]}$, are occupying the queue, the distance to such queue occupations should be at least δ to account for the synchronization error.

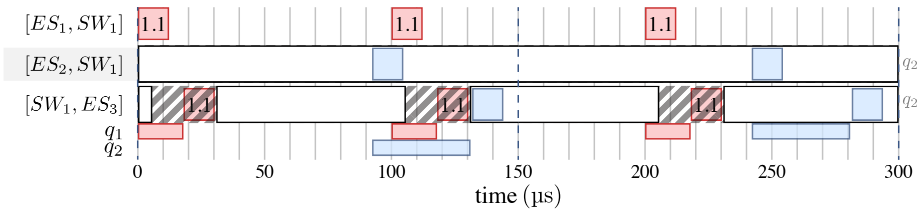
(iii) Similarly, the assigned queue in the receiving egress port $[v_b, v_c]$ is available at time t, and also here there should be a distance of δ to all queue occupations caused by frames arriving from other devices than v_a.

Fig. 13 shows the feasible regions for s_2 of the running example in Sect. 11. In this scenario, s_1 has already been scheduled, i.e., $x = \{s_1\}$. Each of the frames in s_2 must be scheduled without conflicting with s_1. White boxes mark feasible offset intervals, and grayed-out areas depict infeasible intervals. Fig. 13a shows the feasible regions for $f_{2,1}^{[ES_2,SW_1]}$ and $f_{2,1}^{[SW_1,ES_3]}$ when s_2 shares queue q_1 in $[SW_1, ES_3]$ with s_1. Fig. 13b shows the same feasible regions when s_2 is assigned its own queue, q_2.

The figures illustrate how already scheduled frames in x leave holes in the feasible regions for frames that are still waiting to be scheduled. It is apparent that the feasible region of a frame is highly dependent on the availability of its own queue, as well as the availability of the queue it is about to be forwarded into. If frames share the same queue, holes are even larger to prevent flows from being queued at the same time.



(a) Feasible regions using q_1 in egress port $[SW_1, ES_3]$. Notice the δ-sized padding before and after queue occupations.



(b) Feasible regions using q_2 in egress port $[SW_1, ES_3]$.

Figure 13: Feasible region for $f_{2,1}^{[ES_2,SW_1]}$ and $f_{2,1}^{[SW_1,ES_3]}$, when considering the entire hyperperiod for different queue assignments. White boxes mark the feasible intervals.

Condition (i) enforces the *link congestion* property (Sect. 12.1.1). The purpose of condition (ii) and (iii) is related to the *deterministic queue* requirement (Sect. 12.2.3). Condition (ii) ensures that the assigned queue in egress port $[v_a, v_b]$ is available when $f_{i,m}^{[v_a,v_b]}$ is scheduled to leave the queue, and condition (iii) ensures that the queue of egress port $[v_b, v_c]$ is available when $f_{i,m}^{[v_a,v_b]}$ is scheduled to enter.

However, the two conditions are not sufficient to meet the *deterministic queue* requirement. It is not ensured that the queue remains available in the entire interval, from the time the frame enters the queue, until it leaves. Frame $f_{i,m}$ is queued in device v_b in the interval $[f_{i,m}^{[v_a,v_b]}.\phi; f_{i,m}^{[v_b,v_c]}.\phi]$. Consequently, there exist cases where frames are scheduled within their feasible regions, but still result in infeasible schedules. The blue frames of Fig. 13a illustrate such a case, where the two flows use q_1 at the same time. If they use different queues, like in Fig. 13b, then the schedule is feasible. Also note that the feasible regions do not consider other frames of s_i, hence the feasible regions do not prevent that, for instance, two frames of s_i overlap in the time domain.

Scheduling a frame, $f_{i,m}^{[v_a,v_b]}$, means determining the offset, $f_{i,m}^{[v_a,v_b]}.\phi$, within its period. Because a frame is repeated multiple times within the hyperperiod, a valid offset must be in the feasible region in every repetition. Therefore, the feasible region is divided into segments, one for each repetition within the hyperperiod. The notation from Sect. 19.1 is reused to denote the set of repetitions, A, within the hyperperiod:

$$A = \left\{0, 1, \ldots, \frac{\text{hp}}{s_i.T} - 1\right\}, \tag{43}$$

where hp denotes the hyperperiod for all flows in $x \cup \{s_i\}$.

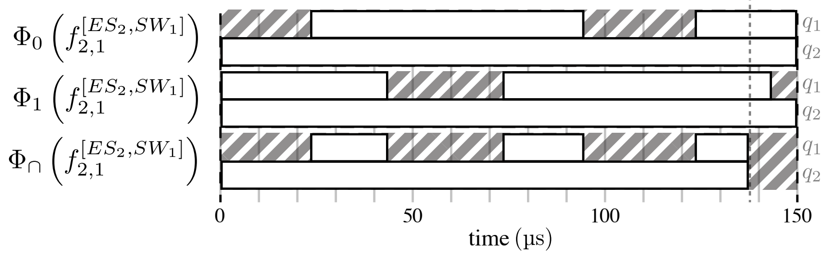
We let $\Phi_\alpha(f_{i,m}^{[v_a,v_b]}, x)$ denote a single segment, $\alpha \in A$, of the feasible region of $f_{i,m}^{[v_a,v_b]}.\phi$, given the partial schedule x. Intersecting all segments yields the subset of intervals present in every segment, i.e., offsets resulting in a feasible solution in every repetition within the hyperperiod. We denote the intersection $\Phi_\cap(f_{i,m}^{[v_a,v_b]}, x)$ and formally define it in Eq. 44.

$$\Phi_\cap\left(f_{i,m}^{[v_a,v_b]}, x\right) = \bigcap_{\alpha \in A} \Phi_\alpha\left(f_{i,m}^{[v_a,v_b]}, x\right) \cap [0, s_i.T - f_{i,m}^{[v_a,v_b]}.L] \tag{44}$$

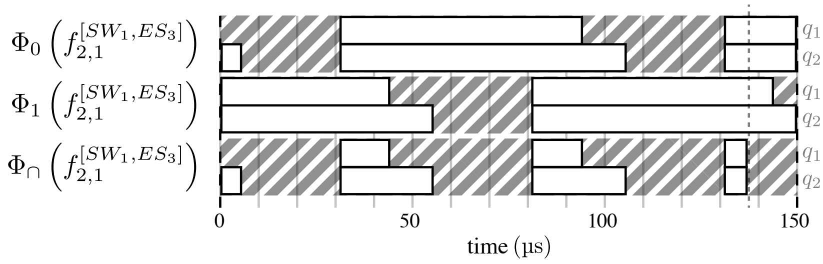
As stated in Sect. 10.2, any frame must be fully transmitted within its period. Hence, an interval corresponding to the duration of the frame has been removed from the end of the feasible region.

Fig. 14 shows the calculation of $\Phi_\cap$ for the feasible regions related to queue assignment q_1 and q_2. The upper half of each row is the feasible region for q_1, and the lower half is for q_2. It shows the feasible region of frames $f_{2,1}^{[ES_2,SW_1]}$ and $f_{2,1}^{[SW_1,ES_3]}$ for the individual segments, as well as the intersected intervals. Because all three frames of flow s_2 have the same duration, the feasible regions are identical for $f_{2,2}$ and $f_{2,3}$.

With the definition of feasible regions, the problem of scheduling s_i reduces to choosing an offset for every frame, $f_{i,m}^{[v_a,v_b]}$, of s_i such that the offset is in $\Phi_\cap(f_{i,m}^{[v_a,v_b]}, x)$, while ensuring that the *deterministic queue* requirement is satisfied, and that the individual frames of s_i do not conflict with each other. Sect. 23.2 presents a strategy for choosing the offsets.



(a) Feasible region for $f_{2,1}^{[ES_2,SW_1]}$.



(b) Feasible region for $f_{2,1}^{[SW_1,ES_3]}$.

Figure 14: Feasible regions for s_2 on q_1 and q_2, when considering individual segments and the intersection.

23.2 As-Soon-As-Possible Strategy

The idea behind the ASAP strategy is to maintain a lower and upper bound, denoted $\underline{\phi}$ and $\overline{\phi}$, respectively, for the offsets of the individual frames of s_i. The bounds are maintained in such a way that feasibility is guaranteed, if offsets are chosen from the feasible regions while satisfying the bounds.

The lower bound is based on an auxiliary function defined in Eq. 45, where $f_{i,m}^{[v_a,v_b]}.(\phi + L)$ is a shorthand notation for $f_{i,m}^{[v_a,v_b]}.\phi + f_{i,m}^{[v_a,v_b]}.L$.

$$\underline{\phi}\left(f_{i,m}^{[v_a,v_b]}\right) = \begin{cases} 0, & \text{if } m = 1 \wedge [v_a, v_b] = s_i.s \\ \lceil f_{i,m-1}^{[v_a,v_b]}.(\phi + L)\rceil, & \text{if } m \geq 2 \wedge [v_a, v_b] = s_i.s \\ \lceil f_{i,m}^{[v_x,v_a]}.(\phi + L) + \delta\rceil, & \text{if } m = 1 \wedge [v_a, v_b] \neq s_i.s \\ \max\{\lceil f_{i,m-1}^{[v_a,v_b]}.(\phi + L)\rceil, \lceil f_{i,m}^{[v_x,v_a]}.(\phi + L) + \delta\rceil\}, & \text{o/w} \end{cases} \tag{45}$$

The function computes a lower bound on $f_{i,m}^{[v_a,v_b]}$, such that the *flow transmission*

Algorithm 2 As-soon-as-possible scheduling of frames of a single flow.

```
1:  function ScheduleFlow($s_i$, $x$)
2:      for $m = 1, 2, \ldots, s_i.k$ do
3:          $f_{i,m}^{[v_a,v_b]}.\underline{\phi} \leftarrow 0,\ f_{i,m}^{[v_a,v_b]}.\overline{\phi} \leftarrow \infty \quad \forall\, [v_a, v_b] \in s_i.r$
4:          $[v_a, v_b] \leftarrow s_i.s$
5:          repeat
6:              $f_{i,m}^{[v_a,v_b]}.\underline{\phi} \leftarrow \underline{\phi}\left(f_{i,m}^{[v_a,v_b]}\right)$
7:              $\phi \leftarrow$ EarliestOffset($\Phi_{\cap}\left(f_{i,m}^{[v_a,v_b]}, x\right), f_{i,m}^{[v_a,v_b]}.\underline{\phi}$)
8:              if $\phi = \infty$ then                                   ▷ No solution exists
9:                  return false
10:             else if $\phi \leq f_{i,m}^{[v_a,v_b]}.\overline{\phi}$ then                  ▷ Success
11:                 $f_{i,m}^{[v_a,v_b]}.\phi \leftarrow \phi$
12:                 $f_{i,m}^{[v_b,v_c]}.\overline{\phi} \leftarrow$ LatestQueueAvailableTime($[v_b, v_c], x, \phi$)
13:                 $[v_a, v_b] \leftarrow [v_b, v_c]$
14:             else                                          ▷ Upper bound violated
15:                 $f_{i,m}^{[v_x,v_a]}.\underline{\phi} \leftarrow$ EarliestQueueAvailableTime($[v_a, v_b], x, \phi$)
16:                 $[v_a, v_b] \leftarrow [v_x, v_a]$
17:         until $[v_a, v_b] = s_i.t$
18:     return $\lambda_i \leq s_i.D$
```

and *link congestion* properties are satisfied, when considering the previous frame on the same link, $f_{i,m-1}^{[v_a,v_b]}$, and the same frame on the previous link, $f_{i,m}^{[v_x,v_a]}$. It is similar to the recursion of Eq. 24. See Sect. 18.1 for a detailed description of the different cases.

Algorithm 2 presents pseudocode for the ASAP strategy. The k frames on each link of s_i are scheduled in order, one by one (line 2). The same frame is scheduled on all links simultaneously by doing a linear search through the feasible regions of each link, starting with the frame on the first link and ending with the frame on the last link (lines 5–17). Each frame is scheduled at time ϕ, being the earliest offset no less than the lower bound, and within the feasible region (line 7). If it is not possible to find such an offset, i.e., $\phi = \infty$, then the search fails (lines 8-9).

If ϕ is below the upper bound (lines 10-13), it is feasible and $f_{i,m}^{[v_a,v_b]}$ is scheduled at this time (line 11). Furthermore, a new upper bound is set for the offset of the frame on the next link. The upper bound is set to be the latest point in time where the queue in egress port $[v_b, v_c]$ is available and has been continuously

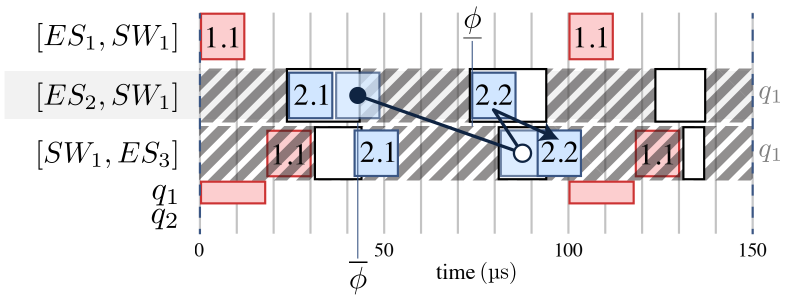


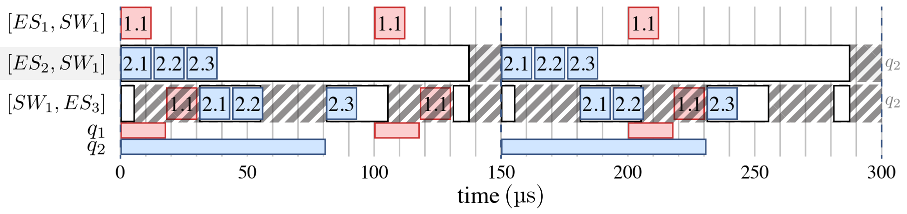
Figure 15: Backtrack-and-reschedule search through feasible regions by means of upper and lower bounds on frame offsets.

since time ϕ (line 12). This prevents queue interleaving between frames of s_i and x in the egress ports. The search then continues with the next link (line 13).

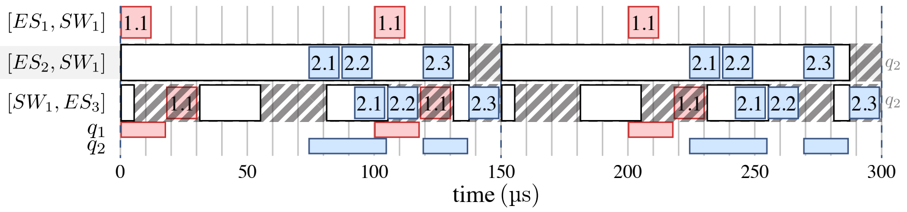
If ϕ violates the upper bound (lines 14-16), it means that it was impossible to schedule the frame when the queue was still available. Consequently, the previous frame must be rescheduled to a later point in time, where — hopefully — the current frame can be scheduled when the queue is available. Thus, the lower bound on the previous frame is increased to the earliest point in time where the queue is available and remains available until time ϕ (line 15). Then, the algorithm backtracks to the previous link (line 16).

Fig. 15 illustrates the principle on the feasible regions for q_1. The figure depicts the scenario, where frame “2.1” has just been scheduled, and the algorithm searches for feasible offsets for “2.2”. The dark circle represents the start of the search, the initial offset for the first frame. As this frame is scheduled, it sets an upper bound, $\overline{\phi}$, on the offset on the next link. The white circle indicates the earliest possible position for the frame on the next link. The offset exceeds the upper bound, which leads to a lower bound, $\underline{\phi}$, on the previous link. The algorithm backtracks to the first link and reschedules this frame to satisfy the new lower bound, which in turn yields a feasible offset on the second link as well.

When all frames of the flow have been scheduled, the algorithm succeeds if the end-to-end latency is within the deadline, and fails otherwise (line 18). The end-to-end latency, λ_i is calculated as in Eq. 28. Sect. 23.3 explains ways of improving the end-to-end latency, thereby increasing the chance of success.



(a) ASAP-scheduling. Objective value: $(1, 37\,\mu s)$.



(b) ALAP-scheduling. Objective value: $(1, 19\,\mu s)$.

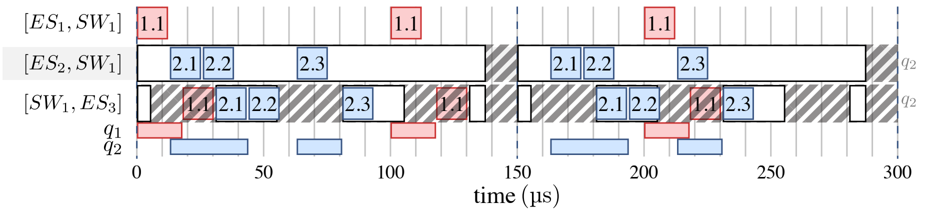
Figure 16: Two different scheduling strategies for s_2 using q_2 in egress port $[SW_1, ES_3]$.

Fig. 15 shows that the algorithm will fail to schedule s_2 using q_1. The third and final frame of s_2 will do a similar backtracking ending up scheduling "2.3" in the last interval of the feasible region for $[ES_2, SW_1]$. However, the frame cannot be scheduled in $[SW_1, ES_3]$, while also satisfying the *flow transmission constraint*. Fig. 16a, on the other hand, shows a successful ASAP-scheduling of flow s_2 using q_2 instead.

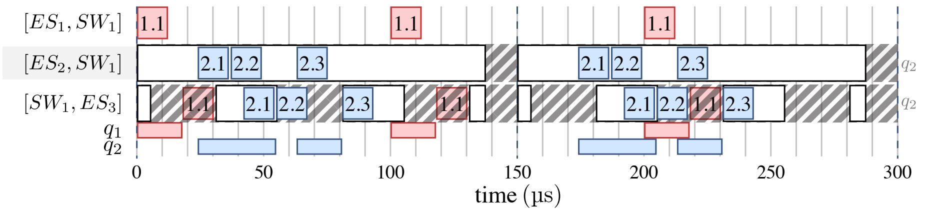
As Fig. 16b illustrates, the same idea can be used in an as-late-as-possible (ALAP) strategy. In such a strategy, the *latest* possible offset is found for each frame in reversed order. The roles of the upper and lower bounds are interchanged, but the basic principle is the same.

23.3 Minimizing End-To-End Latency

The ASAP strategy identifies the earliest feasible offsets for all frames of s_i, for a specific queue assignment. There is, however, no guarantee that the offsets yield the lowest end-to-end latency. The end-to-end latency is improved, if the first frame on the first link is delayed, i.e., moved closer to the last frame on the



(a) Frames moved towards same frame on next link. Objective value: $(1, 24\,\mu s)$.



(b) Frames moved towards last frame on last link. Objective value: $(1, 13\,\mu s)$.

Figure 17: Techniques for minimizing end-to-end latency.

last link. In many cases this is possible by shifting frames left or right within the interval, that they are initially assigned. Such a move does not affect feasibility of the schedule, if the frames of s_i comply with the *flow transmission* and *link congestion* properties with respect to each other.

Fig. 17 presents two methods for minimizing the end-to-end latency, both of which yield better objective values than the original ASAP strategy. Once a frame has been scheduled on all links, the method in Fig. 17a moves frames as close as possible to the same frame on the next link. In this way, queueing delay between the two links is minimized, which in turn reduces end-to-end latency.

The method in Fig. 17b post-processes frame offsets once all frames have been scheduled. The offsets are post-processed such that all frames are moved toward the last frame on the last link. By shifting every frame except the last to the far right, the end-to-end latency is minimized. Analogous methods can be formulated for the ALAP strategy. There are numerous ways of reducing the end-to-end latency, besides the two presented here. These are discussed in Sect. 27.2.

The shifting technique in Fig. 17b yields the optimal objective value for this particular instance. In general, however, it is not obvious which strategy is preferable, and it will vary from instance to instance.

24 Analysis

Sorting all flows can be done in $O(|\mathcal{S}| \log |\mathcal{S}|)$ time, but is negligible compared to the time spend scheduling frames of the individual flows, and is thus disregarded for the rest of the analysis.

Scheduling frames of a flow involves calculating the intersected feasible regions, which is done by removing intervals for each frame that is already scheduled. A hole is added to the feasible region for each repetition within the hyperperiod. The number of frames already scheduled is upper bounded by $|\mathcal{F}|$, and the number of repetitions of each frame within the hyperperiod is upper bounded by $\frac{\text{hp}}{T_{min}}$, where hp is the hyperperiod, and T_{min} is the shortest period. Consequently, the running time for calculating the feasible regions for a frame on all its links is $O(\frac{\text{hp}}{T_{min}}|\mathcal{F}|)$. A tighter bound is achieved by considering the repetitions of frames individually. We denote $|\mathcal{R}|$ the total number of frame repetitions and define it formally in Eq. 46.

$$|\mathcal{R}| = \sum_{s_i^{[v_a,v_b]} \in \mathcal{I}} \frac{\text{hp}}{s_i.T} \cdot |s_i^{[v_a,v_b]}.F| \tag{46}$$

With this notation the running time is $O(|\mathcal{R}|)$.

Every hole added, introduces at most one additional interval by breaking an existing interval into two. Thus, the number of intervals is also upper bounded by $|\mathcal{R}|$. The backtrack-and-reschedule search through the feasible regions considers each interval at most once as Fig. 15 illustrates. Hence, the running time for finding feasible offsets for a frame on all its links is $O(|\mathcal{R}|)$ as well.

In the worst case, every flow instance $s_i^{[v_a,v_b]}$ is assigned once to all $[v_a, v_b].c$ queues, which means that the search for feasible offsets is carried out $[v_a, v_b].c$ times for each of the $s_i.k$ frames of every $s_i^{[v_a,v_b]}$. The maximum number of queues in any egress port is denoted c_{max}. The total number of searches required to schedule s_i is upper bounded by $c_{max} \cdot s_i.k \cdot |s_i.I|$, where $|s_i.I|$ denotes the number of flow instances. The maximum number of queues, c_{max}, may be considered a constant, as it is a result of a hardware limitation and is typically fixed at eight queues. Furthermore, $s_i.k \cdot |s_i.I|$ is equal to the total number of frames in s_i, and thus, summing over all flows gives the total number of frames, $|\mathcal{F}|$. Doing at most $|\mathcal{F}|$ searches, each taking $O(|\mathcal{R}|)$ time, yields an asymptotic running time of $O(|\mathcal{F}| \cdot |\mathcal{R}|)$, which can also be expressed as $O(\frac{\text{hp}}{T_{min}}|\mathcal{F}|^2)$.

The analysis reveals that the running time is highly dependent on the relationship of the periods, as it affects the hyperperiod. For instance, having only a single period will eliminate the $\frac{\text{hp}}{T_{min}}$ factor, but if periods are prime numbers the hyperperiod could be extremely large. The analysis also shows, that the running time is not directly related to topology size, but large topologies often have a large amount of frames to be scheduled.

The main advantage of the heuristic strategy is its polynomial running time, which means that it can produce feasible schedules where the ILP strategy is intractable. There is, however, no guarantee of it finding a solution or the quality of the solution. It makes a polynomial-time, best-effort attempt based on problem specific observations. In spacious schedules, where deadlines are not very tight, and the number of frames per link is reasonable, it is expected to perform and scale well. In very crowded schedules, it might not succeed even if there exists a solution, and the quality could be poor. As presented in Sect. 23.3, there are various techniques for potentially improving the quality of the solution, but one technique is not superior to the others in every scenario.

25 Implementation Details

The heuristic is implemented in Python. To improve performance, computationally expensive modules have been compiled into C with the Cython [40] compiler. The following sections present the two most fundamental details of the implementation.

25.1 Availability Lists

The feasible regions are implemented as linked lists of ordered intervals for every link and queue. The intervals define when the link or queue is available. Initially, the schedule is empty meaning that all links and queues are available at all times. Consequently, the availability lists contain a single interval spanning the entire hyperperiod.

When frames are scheduled on links and in queues, the corresponding intervals are removed from the availability lists. Conceptually, one such block is removed for each time a frame is repeated within the hyperperiod. However, to avoid iterating through the entire hyperperiod an expanding list approach is used instead. The first flow scheduled on a link only removes intervals within its own period. When the next flow is scheduled, the list is expanded to consider the hyperperiod among the two flows, and the current intervals are duplicated to accommodate the new hyperperiod. This continues for all flows on the link. In contrast to considering the global hyperperiod for each list, only the hyperperiod of the flows scheduled on that particular link or queue is considered.

25.2 Feasible Offsets Search

In order to determine offsets for each frame of a flow, the intersections of the feasible regions are calculated based on all availability lists. Each list is compiled into a doubly linked list of intervals, where the corresponding link or queue is available in all repetitions within the hyperperiod of the list. A doubly linked list enables iterating the feasible regions from left in case of ASAP scheduling or from right in case of ALAP scheduling.

Note that the feasible regions in Sect. 23.1 define feasible offsets of frames. In the implementation the intervals specify when the links and queues are available, hence the regions do not depend on the duration of the individual frames. This means that the regions can be reused for all frames of a flow. Instead, the implementation disregards infeasible offsets during the search.

An important difference between the implementation and the description in

Sect. 23.1 is that link and queue intervals are kept separate. This makes the feasible regions search more complicated, because multiple lists are searches simultaneously to find the first offset where both link, sending queue, and receiving queue are available. However, the separation enables the search to identify constraining queues. That is, if an offset is disregarded during the search due to a queue being occupied, then that queue is a constraining queue. Assigning the flow instance to a different queue will potentially allow the frame to start earlier in case of ASAP scheduling or later in case of ALAP scheduling.

26 Metaheuristic Strategy

The term metaheuristic was defined by Glover and Laguna as follows:

> "A meta-heuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality." [41]

Such a master strategy is well-suited for the TT scheduling problem. It may make use of the heuristic approach presented in Sect. 22. The polynomial-time heuristic for scheduling a single flow comes in two variations, an as-soon-as-possible (ASAP) and an as-late-as-possible (ALAP) strategy, each of which can be post-processed in different ways as outlined in Sect. 23.3. As a result, there are numerous modifications of the same heuristic, i.e., different ways of scheduling each flow. Finding the best method for each flow requires full enumeration of all the different ways of combining flows with heuristic variations. The number of combinations grows exponentially with the number of flows, which makes the problem intractable for more than a couple of flows. Hence, the idea is to design a metaheuristic to do a guided search through the different combinations to find a near-optimal solution in a reasonable amount of time.

A fundamental principle in most metaheuristics is a neighborhood search in which similar solutions to the current solution are examined. If an improving solution is found in the neighborhood, the search will continue from this point in the next iteration, in the hope that more high-quality solutions are to be found in this area. This is known as *intensification*. The problem with intensification is that the search tends to get stuck in local optima, which could potentially be very far from the global optimum. Thus, most metaheuristics have techniques for escaping local optima in order to explore the search space on a global scale. This is called *diversification*. The success of a metaheuristic relies on a good balance between intensification and diversification to ensure that the search converges towards the global optimum [42].

Randomization often plays an important role in both the intensification and diversification process, ensuring that different paths are exploited, even if the search ends up where it has previously been. A metaheuristic terminates when a specified stopping criterion is met. It may be based on the execution time, number of iterations, or when there has been no improvement for a significant period of time.

In this section we adapt a well-known metaheuristic to the TT scheduling problem. The chosen metaheuristic is called Greedy Randomized Adaptive Search Procedure (GRASP). The choice of metaheuristic is founded on two main observations made in Sect. 22:

(i) A feasible solution is achievable in a greedy constructive manner, where flows are scheduled one at a time.

(ii) There are many different ways of scheduling the same flow.

These properties make the problem well-suited for the GRASP metaheuristic, which is explained in the following section.

27 Greedy Randomized Adaptive Search Procedure (GRASP)

Each iteration of GRASP [43] consists of two phases: A construction phase, where an initial feasible solution is built, and a search phase, where a neighborhood around the initial solution is examined for improving solutions. The construction phase contributes with diversification, and the local search ensures intensification.

The pseudocode in Algorithm 3 describes the overall strategy of GRASP. As input it takes the problem instance, in this case a set of flows, $\mathcal{S}$, and the parameters, γ and π, which are explained in Sect. 28 and Sect. 29, respectively. Like the heuristic, it outputs a feasible solution x of scheduled flows. The variable x holds the best solution found throughout the search, and is initially empty (line 2). In each iteration, a new solution x' is generated in a greedy fashion (line 4). The solution is subsequently optimized via a local search until reaching a local optimum (line 5). If the new solution results in a better solution than the currently best known, then the best solution is updated (lines 6 and

Algorithm 3 GRASP algorithm

```
1: function GRASP($\mathcal{S}$, $\gamma$, $\pi$)
2:     $x \leftarrow \varnothing$
3:     repeat
4:         $x' \leftarrow$ GreedyRandomizedHeuristic($\mathcal{S}, \gamma$)   ▷ Construction phase
5:         $x' \leftarrow$ LocalSearch($x', \pi$)                          ▷ Search phase
6:         if $f(x') < f(x)$ then
7:             $x \leftarrow x'$
8:     until stop criterion is met
9:     return $x$
```

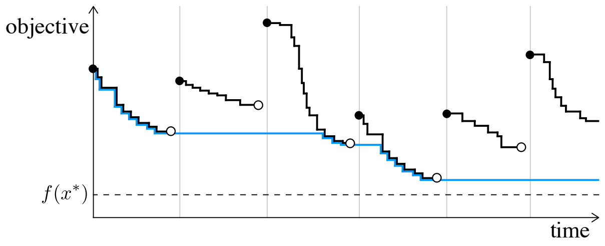


Figure 18: Objective value minimization over time for GRASP.

7). This repeats until the stopping criterion is met (line 8), for instance when the desired execution time has been reached.

Fig. 18 illustrates how the objective value of the current and best known solution progress over time. Black circles indicate the objective value after the construction phase, and white circles indicate the objective value at the end of the local search phase. Vertical lines mark the beginning of a new iteration, the blue line marks the objective value of the best known solution, and the dashed line marks the global optimum.

The subroutines for the construction phase and search phase are highly problem dependent. In the presented metaheuristic strategy, they are both based on the heuristic of Sect. 22, and as such, the metaheuristic can be seen as an optimization layer applied on top of the heuristic. The subroutines are described in detail in Sect. 28 and Sect. 29.

27.1 Objective Function

It should be noted, that there is no guarantee that solution x' is complete, i.e., that all flows are scheduled. When the algorithm terminates we are only interested in complete schedules, and hence the objective value of incomplete schedules should be disregarded. We define the objective function, f, in Eq. 47, as a tuple containing three metrics. The objective function is an extension of the one presented in Sect. 18.

$$f(x) = (|\mathcal{S}| - |x|, \mathrm{K}_x, \Lambda_x) \tag{47}$$

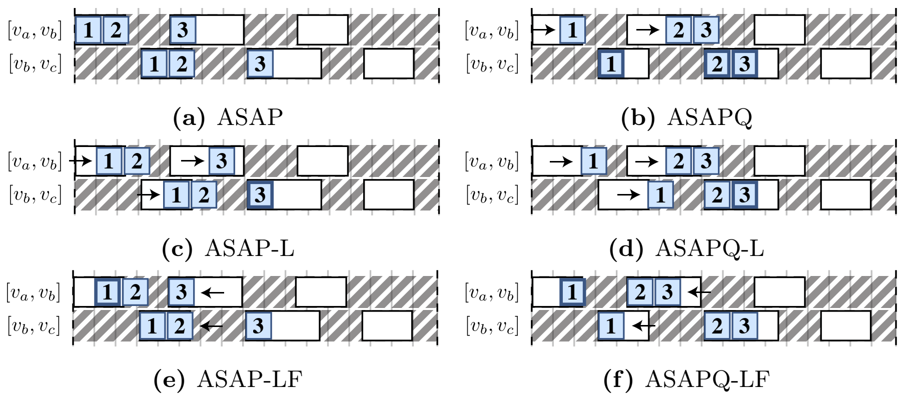
The first metric is the number of flows that remain *unscheduled* in x, hence for all complete schedules the value of this metric is zero. The second and third metric, respectively, are related to queue usage and end-to-end latency of the partial solution x, as elaborated in Sect. 18.

We specify, that one objective value is smaller than another, if the first metric value is smaller. If the values are equal, the second metric is used to break ties, unless these values are also equal, in which case the third and final metric is used for comparison. This definition of $f(x)$ first and foremost encourages finding complete schedules. Queue usage is minimized as a secondary optimization criterion, and lastly, end-to-end latency is minimized.

27.2 Heuristic Variations

As mentioned previously, there are many ways of scheduling a single flow within its feasible regions. In this section we present five variations in additional to the original ASAP heuristic. The same variations exist for ALAP which yields a total of 12 different ways to schedule a single flow.

Fig. 19 shows an example with all ASAP variations, including the original heuristic in Fig. 19a. The sample flow consists of three frames which are scheduled in six different ways on two adjacent links, $[v_a, v_b]$ and $[v_b, v_c]$. The white boxes represent the feasible intervals for the frame offsets, i.e., frames must start in a white box. The primary variation, ASAPQ, is illustrated in Fig. 19b. It attempts to reduce the time frames spend in queues by shifting each frame as close as possible to the same frame on the subsequent link. Frames on the last



(a) ASAP **(b)** ASAPQ **(c)** ASAP-L **(d)** ASAPQ-L **(e)** ASAP-LF **(f)** ASAPQ-LF

Figure 19: ASAP heuristic variations.

are not moved, which is illustrated with a thick border in Fig. 19b. The shift is performed once a frame has been scheduled on all links, affecting the lower bound of the next frame. Consequently, frames are not necessarily assigned the same feasible interval as with the original ASAP variation. In Fig. 19b frame “2” is assigned the middle intervals, whereas the original heuristic assigns it to the first intervals. By reducing the time frames spend in queues, the overall latency is minimized. As an important side effect, the method reduces the overall time the queue is occupied, which could lead to better queue utilization and a lower total number of queues.

The remaining four variations represent different ways of post-processing the offsets once all frames have been assigned feasible intervals by either ASAP or ASAPQ. In ASAP-L (Fig. 19c) all frames are shifted toward the last frame to reduce end-to-end latency. The schedule produced by ASAP-LF (Fig. 19e) has been through an additional post-processing step, where all frames are also shifted toward the first frame. ASAPQ-L and ASAPQ-LF are variations of ASAPQ that have been post-processed in the same two ways.

The last four variations do not require a new search through the feasible regions, and are thus computationally inexpensive. Furthermore, ASAP and ASAPQ make use of the same set of feasible regions, and thus, all six variations arise from searching through the same feasible regions twice, which is more computationally efficient than computing each variation separately.

The exact same variations can be formulated for the ALAP heuristic, but every

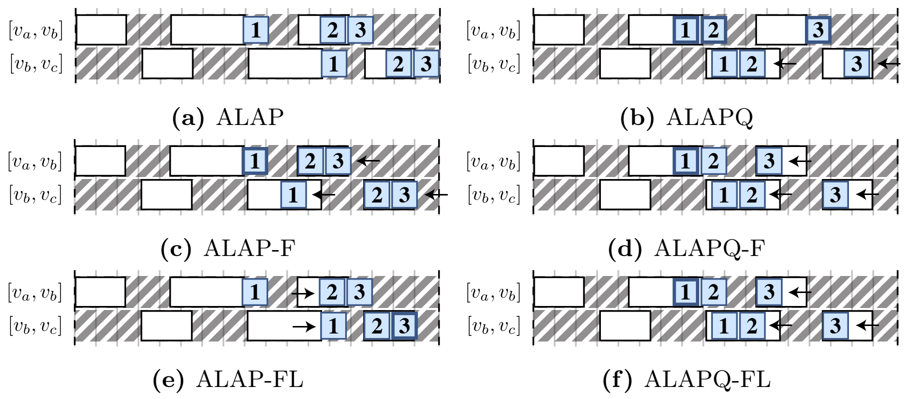


Figure 20: ALAP heuristics variations.

shift is reversed compared to ASAP. Consequently, the post-processing steps first move toward the *first* frame, then the *last*. Fig. 20 depicts the variations for ALAP.

28 Greedy Randomized Heuristic

For the TT scheduling problem we propose a greedy randomized heuristic for the construction phase. The heuristic is an extension of GreedyHeuristic($\mathcal{S}$) presented in Algorithm 1 of Sect. 22. Recall that Algorithm 1 constructively schedules one flow at a time in a fixed order. Each flow is greedily scheduled such that the queue usage is minimized, in the partial schedule of the already scheduled flows.

Randomization is an important aspect of the construction phase to ensure diversification. The idea behind the construction phase is to generate unique starting points for the local search of each iteration as depicted in Fig. 18. If the same solution is constructed in each iteration, the construction phase has no effect. Algorithm 1 deterministically uses the same heuristic for every flow, which makes it unsuited as a construction phase heuristic. Furthermore, it applies the same scheduling variation to all flows, but the purpose of the metaheuristic is to find — for each individual flow — the best variation among the available.

Consequently, we propose an extended randomized version, which we denote GreedyRandomizedHeuristic($\mathcal{S}, \gamma$). The algorithm is modified to handle each flow in the following ways:

- The increase in objective value is predicted for each of the twelve heuristic variations.
- A Restricted Candidate List (RCL) of the γ best heuristic variations is maintained, i.e., the variations yielding the *lowest* objective value increase.
- The flow is scheduled with a random method from RCL.

The length of RCL, is an input parameter, γ, which should be tuned to get the best results. If $\gamma = 12$, the algorithm simply chooses a random variation, and if $\gamma = 1$ the best variation is always chosen. Note, however, that several variations may yield the same minimum increase in objective value, and thus $\gamma = 1$ still represents a random choice between these equally good options.

If a method fails to schedule the flow, it is not added to RCL, as this would lead to an incomplete solution. Thus, it could be the case that all twelve variations fail, in which case the algorithm terminates and returns the partial solution constructed so far.

The increase in objective value is predicted by computing frame offsets and queue assignments for each heuristic variation, but without adding the flow to the partial solution x. If new queues are utilized, that are not already used by flows in x, then the queue usage metric increases. The end-to-end latency metric increases by the latency of the flow being scheduled.

29 Local Search

The purpose of the local search phase is to intensify the search by investigating a well-defined neighborhood of solutions similar to the current solution. For the TT scheduling problem this corresponds to schedules where the majority of flows are scheduled exactly as in the current solution. It is likely that a better solution arises from rescheduling only a couple of flows. The local search attempts to identify such reschedulings by destroying part of the schedule, and repairing it in a different way.

We initially define the neighborhood of a partial schedule x as follows: All the schedules which can be constructed by removing up to π flows from x, and subsequently rescheduling them using one of the twelve heuristic variations presented in Sect. 27.2. With this definition, the size of the neighborhood grows exponentially with π. Consequently, three measures are taken to limit the neighborhood size:

- The maximum number of flows to remove is fixed at a low value, for instance $\pi \leq 5$.
- Only flow combinations sharing at least one link are considered for removal. Rescheduling unrelated flows is unlikely to improve the solution quality by much.
- Each flow can be scheduled in twelve different ways yielding 12^π different repair permutations. Instead of enumerating them all, the strategy from Sect. 28 is reused to greedily repair flows one by one.

The following section presents two different strategies for searching through the reduced neighborhood.

29.1 Search Strategies

The local search is a metaheuristic itself, typically implemented as a *hill climbing* algorithm, only accepting improving solutions. It comes in two variations [44]: *First ascent* hill climbing, which accepts the first improving solution from the neighborhood, and *steepest ascent* hill climbing, where the entire neighborhood is enumerate and the most improving solution is accepted. The local search terminates when the entire neighborhood has been searched without finding an improving solution. In this case, the current solution is a local optimum with respect to the defined neighborhood.

Algorithm 4 Local search using first ascent hill climbing

```
1: function LocalSearch($x$, $\pi$)
2:     repeat
3:         $x^c \leftarrow x$
4:         for $y \in D_\pi(x^c)$ do
5:             $x' \leftarrow x^c - y$                                  ▷ Destroy
6:             for $s_i \in y$ do
7:                 ScheduleFlowGreedyRandomized($s_i, x'$)       ▷ Repair
8:             if $f(x') < f(x^c)$ then
9:                 $x \leftarrow x'$
10:                break
11:    until stop criterion is met $\vee\ x^c = x$
12:    return $x$
```

For large neighborhoods, full enumeration might be too time consuming, in which case a stopping criterion may be defined in terms of maximum iteration count or limited execution time. The stopping criterion ensures that GRASP returns to the construction phase. As mentioned, a good metaheuristic implementation has a proper balance between intensification and diversification. For GRASP, this means spending the right amount of time in the construction phase versus the local search phase. Hence, the stopping criterion for the local search phase is an important parameter which should be experimentally tuned to get the best results.

Algorithm 4 shows pseudocode for a first ascent variation of the local search. The notation $D_\pi(x^c)$ is used to denote the set of all possible combinations of choosing at most π flows from the current solution x^c, such that all the chosen flows share at least one link. These are the set of flows considered for deletion. Hence, line 4 iterates over all the different ways to delete flows from x^c. Flows are removed to create the incomplete solution, x' (line 5), which is repaired by rescheduling flows in a greedy randomized fashion.

Once an improving solutions has been found, the neighborhood loop is terminated, and a new local search iteration begins starting from the new and improved x (line 3). The local search terminates when the stop criterion is met, or the entire neighborhood of the current solution has been searched without finding an improving solution.

Converting the search strategy into a steepest ascent version is straightforward.

It involves maintaining the best x' found so far, and only update x once the entire neighborhood has been searched. Apart from this, the procedure is the same.

30 Analysis

Recall from Sect. 24 that each frame is scheduled on all its links in time $O(\frac{\text{hp}}{T_{min}}|\mathcal{F}|)$. The construction phase reschedules each frame four times, once for each of the strategies ASAP, ASAPQ, ALAP, and ALAPQ. The post-processing steps resulting in a total of twelve variations are calculated in constant time for each frame. Hence, the total running time for the greedy randomized heuristic is $O(s \cdot \frac{\text{hp}}{T_{min}}|\mathcal{F}|^2)$, where s is the number of heuristic variations that require their own feasible regions search, in this case four.

The local search utilizes the same scheduling algorithm, but reschedules at most π flows. Let Π denote the maximum number of frames removed in any iteration. Then rescheduling all removed frames takes $O(s \cdot \Pi \cdot |\frac{\text{hp}}{T_{min}}|\mathcal{F}|)$ time. In a naive implementation, flows are removed by reconstructing the entire schedule without the removed flows. This takes $O(\frac{\text{hp}}{T_{min}}|\mathcal{F}|^2)$ time. With clever bookkeeping the deletion can be optimized by removing the Π frames from the current schedule. For metaheuristics in general, this is known as *delta evaluation.* Sect. 31.2 discusses how to do this for the TT scheduling problem.

Overall, each iteration of the GRASP strategy runs in polynomial time, and it only improves solution quality as it progresses. Hence, it fills the gap between the exponential-time ILP strategy, and the polynomial-time heuristic strategy. It is expected to perform well on test cases that are too large for the ILP strategy, and too congested to solve using a single heuristic strategy. For test cases that can be solved using a single heuristic strategy, GRASP works as an optimization layer minimizing the objective function.

The disadvantage of the metaheuristic is the number of settings to tune. The performance of GRASP is highly dependent on the parameter values, the neighborhood definition, and the local search strategy. Determining the best settings requires problem-specific knowledge and experimental evaluations.

31 Implementation Details

Like the heuristic strategy, the metaheuristic is implemented in Python with selected modules compiled into C with Cython. The following sections present the key details related to the implementation.

31.1 Post-processing

The presented post-processing steps involve moving frames within the interval they are initially assigned. As long as the move is only done within the assigned interval it can never violate constraints related to other flows. However, the move could violate constraints related to other frames of the same flow. In particular, two properties must be respected:

(i) The link congestion property, i.e., frames on a link cannot overlap.

(ii) The flow transmission property, i.e., there should be a spacing of δ between the same frame on two adjacent links.

This is implemented by traversing the frames in the right order, maintaining upper and lower bounds. Consider the post-processing step that moves all flows toward the last frame on the last link. See Fig. 19c. By definition, this frame does not move, and it sets the upper bound for the offsets of the other frames, such that (i) and (ii) are satisfied.

The route of the last link is traversed in reversed order, starting from the second-to-last link. Each frame is moved as far to the right as possible while respecting the upper bounds. Then, a new upper bound is set for the previous link. This repeats until the frame has been moved on the very first link. After that, the post-processing step considers the second-to-last frame on the last link. It is moved to the right while respecting (i), and then the procedure continues for the remaining links. The post-processing finishes when the first frame on the first link has been moved towards its upper bound. The length of this move is equal to the reduction in end-to-end latency. The procedures for the other post-processing steps are equivalent.

31.2 Deleting flows

Flows are deleted from the current solution by creating a copy where the deleted flows are left out. As the partial schedule is copied, the new objective value is calculated. Hence, the implementation does *not* feature delta evaluation. A copy is created in every iteration of the local search. However, within an iteration, the same copy is used for trying out all heuristic variations, and for repairing all of the removed flows.

In order to implement true delta evaluation, additional data structures are needed in order to keep track of the effects of each flow. For instance, when flow s_i is removed the availability lists (see Sect. 25.1) must be updated, such that the links and queues are made available in the intervals previously occupied by the frames of s_i. This requires bookkeeping of all the intervals that were removed due to the scheduling of s_i, taking into account that the availability lists may have expanded since s_i was scheduled.

Another important aspect of delta evaluation is determining the objective value of the new partial schedule. When s_i is deleted, it is straightforward to subtract the end-to-end latency of s_i from Λ. However, determining the impact on K is more involved. K is reduced only if the queue is left unused, i.e., no other flow is assigned that queue. It requires maintaining the assignments of flows to queues and updating them accordingly when flows are deleted and repaired.

32 Experimental Evaluation

In this section, we evaluate the performance of each of the three presented strategies. The strategies are evaluated in terms of running time and solution quality. The number of egress port queues used by a solution is of particular interest and is considered the primary optimization objective. The strategies are applied to synthetic, industrial-sized test cases, with high link utilization to provoke the use of multiple queues.

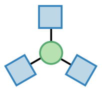
All experiments were conducted on the DTU High Performance Computing Cluster, which enables performing up to one hundred experiments in parallel. 16 GB of RAM were reserved for each job. Due to the cluster design, not all jobs were executed on the same hardware. However, the majority of jobs ran on a 2.80 GHz Intel Xeon E5-2680 processor.

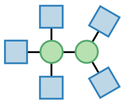
33 Test Cases

Generating interesting test cases is a problem in itself. If there is too much space in a schedule, every flow is trivially scheduled with low end-to-end latency and using only a single queue in each egress port. However, if a schedule is too congested, there may not exist a feasible solution, and hence, the scheduling problem is unsolvable.

We assume some parameters are fixed at constant values. For instance, the network precision is assumed to be $\delta = 5.008\,\mu s$. The transmission rate for all links is fixed at 1 Gbps, and the propagation delay of each link is assumed negligible, i.e., it is set to zero. Every egress port is assumed to have eight queues. Furthermore, the deadline of each flow is set to its period, i.e., there is no explicit deadline definition. Instead all frames must start and finish within their period.

We construct six topologies of varying size. The topologies are industrial sized, and are derived from the work presented in [45]. The topologies are grouped into three categories based on their size. There are three small topologies, two


(a) $|\mathbf{SW}| = 1, |\mathbf{ES}| = 3$


(b) $|\mathbf{SW}| = 2, |\mathbf{ES}| = 5$

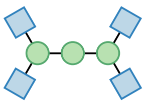
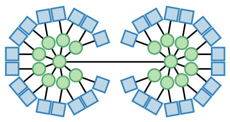

(c) $|\mathbf{SW}| = 3, |\mathbf{ES}| = 4$

Figure 21: Small network topologies.


(a) $|\mathbf{SW}| = 18, |\mathbf{ES}| = 32$

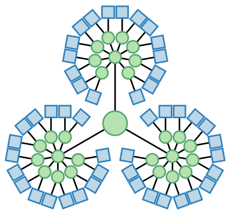

(b) $|\mathbf{SW}| = 28, |\mathbf{ES}| = 48$

Figure 22: Medium network topologies.

medium, and a single large, shown in Fig. 21, Fig. 22, and Fig. 23, respectively. Blue squares illustrate end systems and green circles illustrate switches. The figure captions present the number of switches and end systems in each topology. Note, that there are no redundant routes in the topologies, i.e., there exists only one route between any pair of end systems.

33.1 Flows

The hyperperiod of all flows defines the width of the schedule, and has a major impact on the complexity of the problem. Thus, the hyperperiod is an important aspect to consider, when evaluating performance. We define three hyperperiods of 1 ms, 6 ms, and 30 ms. For each choice of hyperperiod we define high-rate and low-rate periods as defined in Table 8. High-rate flows have a data size of either one, two, or three times the Maximum Transmission Unit (MTU) of 1500 bytes. Low-rate flows have data sizes 10, 20, 40, 60, or 100 times MTU. The choice of periods and data sizes are inspired by [26].

In order to generate flows, that yield difficult scheduling problems in terms of queue usage and end-to-end latencies, the link utilization should be relatively high. Thus, we propose the following procedure for generating applications with

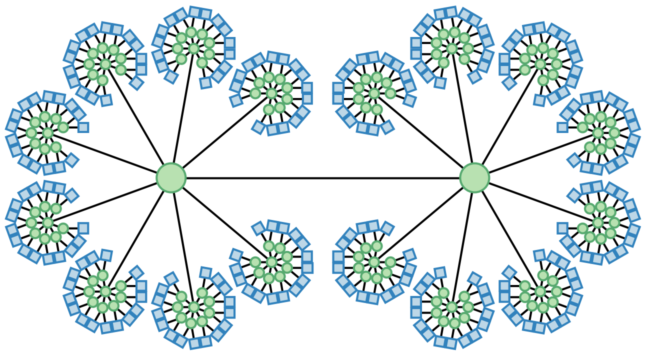


Figure 23: Large network topology with $|\mathbf{SW}| = 146, |\mathbf{ES}| = 256$.

high link utilization: High-rate and low-rate flows are repeatedly added to the set of flows. The sending and receiving end system are chosen at random among the end systems in the topology. The flow periods are randomly chosen based on Table 8, and the data size is chosen from the set of data sizes mentioned above. This procedure is repeated until meeting a threshold, u, for the average link utilization on all links in the topology, or until no flow can be added without exceeding the threshold by 20% on one of its links. When the procedure terminates all links have roughly the same utilization with an average close to u. By incrementally increasing u in the interval between 5% and 45%, flows are added to the network until multi-queue scenarios arise.

The interesting values for u differ for each choice of topology and hyperperiod. Low values yield too easy instances, and high values yield instances that are impossible to schedule. Hence, to determine the proper values of u, the ASAP heuristic is used as a proxy function to indicate when the network is sufficiently saturated with flows.

For each choice of topology and hyperperiod, we generate three classes of link utilization. The classes are denoted *high utilization*, *medium utilization*, and *low utilization*. They are based on the objective value obtained by the ASAP heuristic. The *high* class contains the 30 observations with the highest queue usage. The *low* class consists of the 30 observations with the highest end-to-end latency, that use no excess queues, and *medium* is the 30 median observations with respect to both queue usage and end-to-end latency. In total 1620 test cases are generated in this way, 270 for each of the six topologies.

Hyperperiod	**High-rate periods**	**Low-rate periods**
1 ms	100 µs, 200 µs, 500 µs	1 ms
6 ms	100 µs, 150 µs, 200 µs, 500 µs	1 ms, 2 ms, 6 ms
30 ms	100 µs, 150 µs, 200 µs, 300 µs, 500 µs	5 ms, 10 ms, 30 ms

Table 8: High- and low-rate periods for specific hyperperiods.

		topology size		
		small	medium	large
hyperperiod	1 ms	$(14, 168)$	$(55, 490)$	$(299, 1749)$
	6 ms	$(13, 429)$	$(49, 1076)$	$(257, 3672)$
	30 ms	$(15, 692)$	$(59, 2980)$	$(316, 16449)$

Table 9: Application size for different test case parameters. Each cell contains a tuple $(|\overline{\mathcal{S}}|, |\overline{\mathcal{F}}|)$, denoting the average number of flows and frames, respectively.

Table 9 shows the average number of flows and frames for every pair of topology class and hyperperiod. Overall, the test instances range from a few hundred frames to tens of thousands of frames.

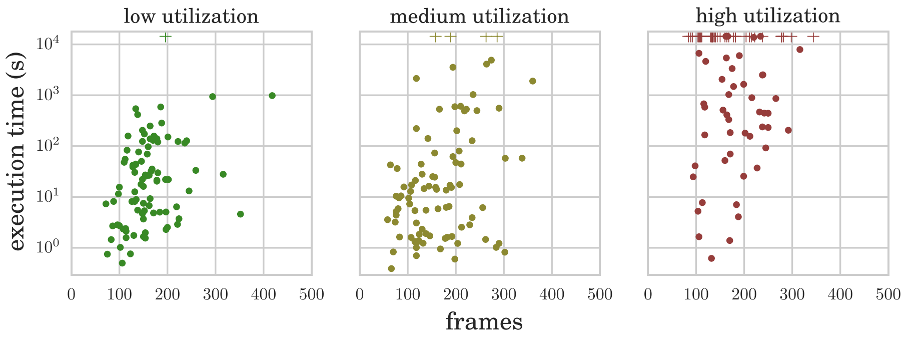
34 ILP Strategy

We propose two variations of the ILP strategy of Sect. 17. The difference between the two is only in the objective function definition. Recall that the general form of the objective function is

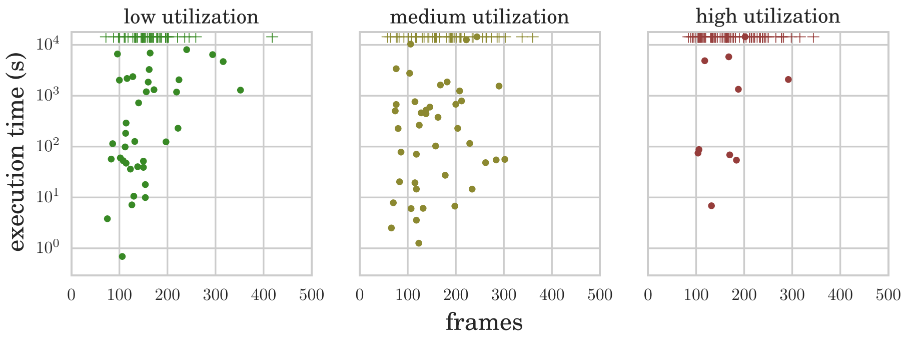
$$z = c_1 \cdot \mathrm{K} + c_2 \cdot \Lambda \tag{48}$$

where K is the number of excess queues used by TT flows, and Λ is the total end-to-end latency introduced as a result of interference between flows.

The first variation, denoted ILP-K, only minimizes the number of queues, corresponding to $c_1 = 1, c_2 = 0$. The second variation, denoted ILP-KΛ, has K as its primary optimization objective, and Λ as a secondary objective. This corresponds to setting $c_1 = M$ and $c_2 = 1$, where M is a sufficiently large number, for instance the sum of all periods.



(a) ILP-K



(b) ILP-KΛ

Figure 24: Execution time of ILP strategy for different levels of link utilization. Points indicate a proven optimal solution, + indicate the best known solution when exceeding the four-hour time limit.

The strategies are benchmarked on the smallest topology size and the shortest hyperperiod (1 ms). There are 270 such test cases. A cut-off time of four hours is used, after which the algorithm terminates, returning the best known solution. Thus, if the time limit is exceeded there is no proof of optimality, even though the best known solution may be optimal or near-optimal.

Fig. 24 shows the execution time for each of the 270 test cases, categorized by link utilization. The execution time is plotted on a logarithmic scale to visualize the exponential nature of the problem. The figure shows that ILP-K is in general easier to solve, as more executions finish within the time limit. This seems reasonable as ILP-KΛ solves a multi-objective problem. The plots also reveal that the problem complexity grows with the link utilization. Fewer executions are solved to optimality as the link utilization increases. As a result only 53% of the high utilization test cases are solved to optimality for ILP-K, and only 11% for ILP-KΛ.

Both ILP strategies seem unsuited for anything but the smallest topologies and shortest hyperperiods. They can, however, serve as important tools for benchmarking other strategies, if test cases can be solved to optimality in a reasonable amount of time.

35 Heuristic Strategy

Sect. 22 presents a polynomial-time heuristic. In Sect. 27.2 the heuristic is generalized into twelve different variations. In the following sections, we evaluate if one heuristic variation is preferable to the others, how solution quality compares to the optimum, and finally how an increase in problem size affects running time.

35.1 Variations Comparison

There are four different heuristics, ASAP, ASAPQ, ALAP, and ALAPQ, each of which can be post-processed in two ways in addition to the original heuristic. This yields twelve variations. None of the heuristics guarantee to find a solution if one exists. Hence, it is an important metric to evaluate how many test cases are solved by each heuristic.

Fig. 25 shows the percentage of solved test cases for each heuristic variation. In case the heuristic is not able to schedule all flows, it will return a partial schedule with the flows it was able to schedule. Here, however, we only consider a problem solved if all flows are scheduled.

Recall, that the test cases have been constructed using ASAP as a proxy function. Thus, per definition ASAP solves all test cases. From the figure, the ASAP strategies seem slightly better at finding solutions than the ALAP strategies.

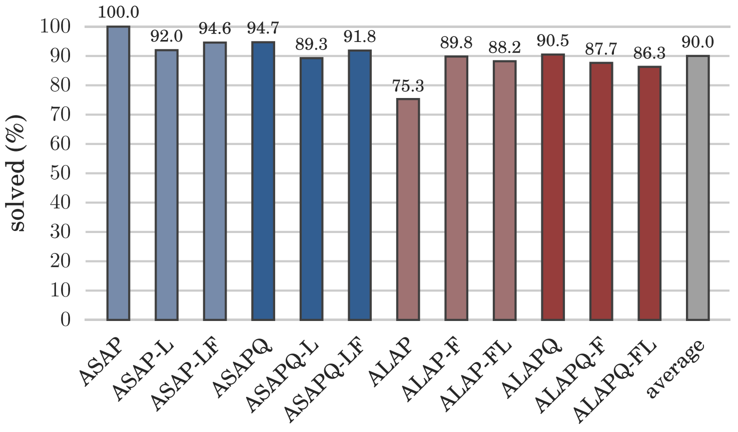


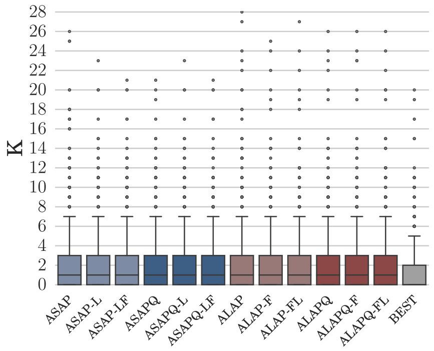
Figure 25: Percentage of solved test cases for heuristic variations.

This could be a result of the construction procedure. That is, ASAP strategies are given an unfair advantage when test cases are constructed. With this in mind, we calculate the average solve percentage in the last column of Fig. 25. It is expected that the heuristics will achieve around 90% solved test cases in real-life scenarios with similar link utilizations, topology sizes, etc.

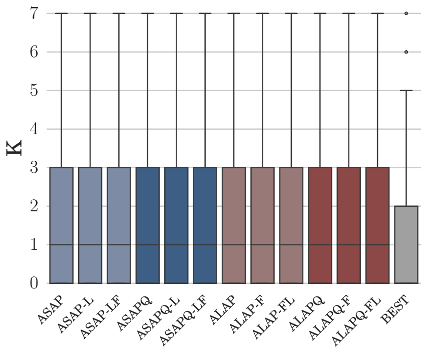
1134 of 1620 test cases are solved by *all* variations. This subset of test cases is used to compare the solution quality among the heuristic variations. Fig. 26 and Fig. 27 show box plots for the distributions of test cases in terms of queue usage (K) and end-to-end latencies (Λ), respectively.

A box plot [46] is a standardized way of visualization distributions of data based on the five metrics: Minimum, first quartile, median, third quartile, and maximum. The rectangle spans the first quartile, median, and third quartile. The height of the rectangle is denoted the Interquartile Range (IQR). The maximum is defined as the observation that is $\frac{3}{2}$IQR above the third quartile, and the minimum is conversely the observation $\frac{3}{2}$IQR below the first quartile. Observations above the maximum or below the minimum are considered outliers and are depicted as individual points.

For instance, the box plots in Fig. 26 depict that 75% of all test cases are solved using 3 of fewer excess queues, and 50% are solved using zero or one excess queue. The box plots are identical for all twelve variations when considering

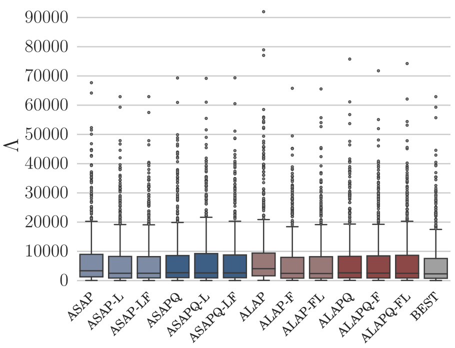


(a) box plot

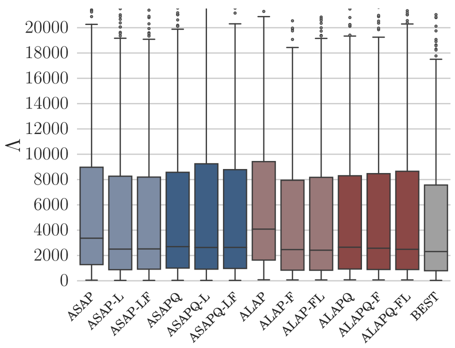


(b) box plot, without outliers

Figure 26: Queue usage comparison for heuristic variations.



(a) box plot



(b) box plot, without outliers

Figure 27: End-to-end latency comparison for heuristic variations.

queue usage. For end-to-end latency there is a little more variation, but in general the heuristics perform similar.

Fig. 28 shows queue usage categorized by link utilization. As expected, test cases with higher link utilization require more queues. The ALAP strategies yield slightly better queue usage for medium and high utilizations. However, when considering both queue usage and end-to-end latencies for varying link utilizations, hyperperiods, and topology sizes, it is not possible to determine a superior strategy among the twelve. See Fig. 34 and 35 in Sect. 39.

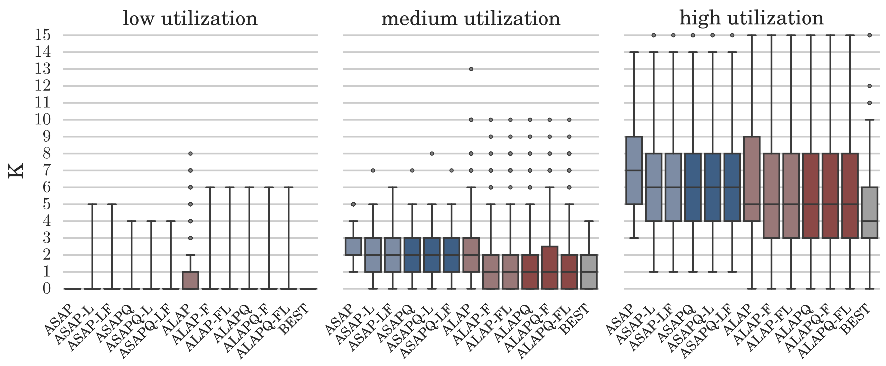


Figure 28: Queue usage for heuristic variations at different link utilizations.

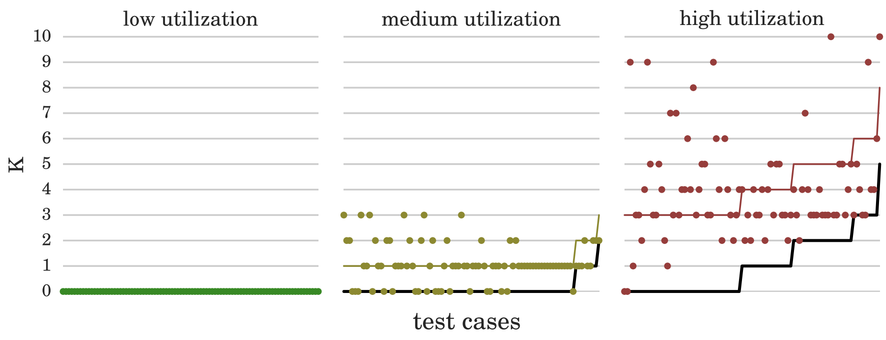


Figure 29: Queue usage for BEST heuristic compared with optimum.

As a result, it seems like a reasonable strategy to perform all twelve heuristics on each test case and choose the one yielding the lowest objective value. The distributions for this strategy, denoted BEST, are depicted in the last column of all box plots. BEST is used for comparison with the metaheuristic in Sect. 36.4.

35.2 Compared to Optimum

Fig. 29 shows queue usage for the smallest three topologies with hyperperiod 1 ms. These are the test cases solved by ILP-K, hence, the optimum is known. If ILP-K failed to find the optimum within the time limit, the best feasible solution is assumed the optimum. Black lines depict the optimal number of

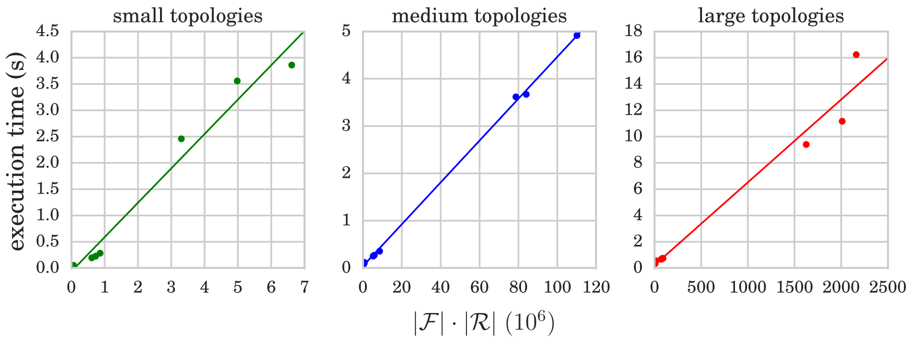


Figure 30: Average heuristic execution time as a function of $|\mathcal{F}| \cdot |\mathcal{R}|$.

excess queues, and points represent the best solution found using any of the twelve variations, i.e., the BEST heuristic. The thin colored lines depict the average number of extra queues used by the heuristic compared to the optimum. For medium utilization, the heuristic on average used 1.0 more queues than the optimal solution. For high utilization the same number is 3.0. For low utilization the number is zero as ASAP solves all these with no excess queues per definition. The heuristic found an optimal solution for 28% of the medium utilization test cases, and for 7% of the high utilization test cases.

35.3 Execution Time

In Sect. 24 the asymptotic running time of the heuristic is analyzed to be $O(|\mathcal{F}| \cdot |\mathcal{R}|)$, where $|\mathcal{F}|$ is the total number of frames, and $|\mathcal{R}|$ is the total number of frame repetitions. This is due to the fact that each of the $|\mathcal{F}|$ must be scheduled while considering a partial schedule consisting of $|\mathcal{R}|$ already occupied intervals. Hence, the running time of the algorithm grows with not only the number of frames, but also the number of repetitions of individual frames.

The running time is visualized as a function of $|\mathcal{F}| \cdot |\mathcal{R}|$ in Fig. 30. The figure indicates a linear relationship which is in accordance with the analysis. Each data point represents the average performance of all heuristics for a specific combination of hyperperiod and link utilization.

Deviations from the straight line are most likely caused by multi-queue scenarios. That is, if a flow cannot be scheduled on a queue, all the work done to schedule it on this particular queue is discarded, and the flow is attempted scheduled on the next queue. Hence, test cases where this happens often take longer than test cases that are immediately scheduled on the first queue. This is not accounted for in Fig. 30.

36 Metaheuristic Strategy

Sect. 26 presents a metaheuristic strategy, which basically searches for the best way of combining the twelve heuristics. In the following subsections we decide on a stopping criterion for the metaheuristic, tune its parameters for better results, and finally compare the performance of the tuned metaheuristic with the optimum and with the heuristic approach.

36.1 Execution Time

The stopping criterion for the metaheuristic is defined as a maximum execution time. The objective value will only improve over time, and thus, higher execution times will result in better solutions. The objective value decreases the most in early iterations, and at the later stages improvements are small and unlikely. Hence, at some point, the additional execution time is not worth the potentially small decrease in objective value. The execution time should be defined based on this.

Furthermore, the execution time should take into account the size of the test case, as each iteration takes longer for larger test cases. As a result, we decide to let the smallest test cases run for one minute, and the largest for one hour. Running times for the remaining test cases increase with the hyperperiod and topology size as shown in Table 10.

The variable execution times allow all test cases to roughly explore the same fraction of the solution space, while still ensuring that the algorithm terminates

		topology size		
		small	medium	large
hyperperiod	1 ms	1 min	5 min	15 min
	6 ms	5 min	15 min	30 min
	30 ms	15 min	30 min	60 min

Table 10: Fixed execution times for GRASP.

within a reasonable amount of time. The execution times represent a trade-off between solution quality and tractability.

36.2 Parameter Tuning

The GRASP metaheuristic as described in Sect. 26 comes with several parameters listed below.

(i) The length, denoted γ, of the Restricted Candidate List (RCL).

(ii) The ratio of time spent in the local search phase compared to the construction phase. This parameter is denoted l.

(iii) The neighborhood definition in terms of the maximum number of flows to remove from the current solution. This parameter is denoted π.

(iv) The local search strategy, denoted a. Either a steepest ascent or a first ascent hill climbing metaheuristic.

The parameters should be tuned to find the values yielding the best results. The set of test cases is divided into two subsets, a training set and a test set. The training set consists of 540 (33%) representative test cases, and the test set is the remaining 1080 test cases. The parameter values are tuned on the training set. Subsequently, the metaheuristic is benchmarked on the test set with fixed parameter values.

	$\gamma = 1$	$\gamma = 2$	$\gamma = 4$	$\gamma = 6$
solved	90.1%	89.6%	89.3%	87.5%
objective (avg)	$(1.89, 5154)$	$(1.95, 5370)$	$(2.03, 5450)$	$(2.20, 5653)$
objective (std)	$(2.43, 7683)$	$(2.53, 8063)$	$(2.70, 7958)$	$(2.88, 8230)$

Table 11: Performance of training set for varying γ.

	$l = 2$	$l = 3$	$l = 5$	$l = 10$
solved	98.7%	98.7%	98.7%	98.7%
objective (avg)	$(0.94, 4938)$	$(0.96, 4870)$	$(0.96, 4838)$	$(0.97, 4906)$
objective (std)	$(1.58, 8193)$	$(1.65, 8003)$	$(1.65, 7611)$	$(1.76, 8066)$

Table 12: Performance of varying l on training set. Parameters fixed: $\gamma = 1$.

36.2.1 Construction Phase

The parameter, γ, relates only to the construction phase of the metaheuristic. Thus, it can be determined independently of the others. Table 11 shows the benchmark results for four different values of γ. Each of the 540 test cases have been solved 10 times with the GreedyRandomizedHeuristic as explained in Sect. 28.

The performance of each parameter value is evaluated based on three features: The percentage of solved test cases, the average objective value (avg), and the standard deviation (std) of the objective value. Naturally, a good parameter value yields a high solve percentage, and has a low average objective value with little deviation. The table shows that $\gamma = 1$ is preferable on all three features. Hence, from this point on, γ is fixed at 1. Recall that $\gamma = 1$ corresponds to choosing randomly among the heuristic variations that yield the minimal objective value for each flow.

36.2.2 Local Search Phase

Three parameters are related to the local search phase: Time spent in the local search phase, neighborhood definition, and local search strategy. Ideally, all combinations of all parameter values should be tried to find the optimal settings.

	$\pi = 2$	$\pi = 3$	$\pi = 4$	$\pi = 5$
solved	98.7%	98.7%	98.7%	98.7%
objective (avg)	$(0.92, 4848)$	$(0.94, 4938)$	$(0.91, 4772)$	$(0.92, 4915)$
objective (std)	$(1.53, 7850)$	$(1.58, 8193)$	$(1.54, 7550)$	$(1.59, 7922)$

Table 13: Performance of varying π. Parameters fixed: $\gamma = 1$, $l = 2$.

	$a =$ first ascent	$a =$ steepest ascent
solved	98.5%	98.7%
objective (avg)	$(0.91, 4805)$	$(0.91, 4772)$
objective (std)	$(1.57, 7618)$	$(1.54, 7550)$

Table 14: Performance of search strategies with $\gamma = 1$, $l = 2$, $\pi = 4$.

However, to limit the number of combinations to evaluate, the parameter values are instead determined greedily, ordered by the assumed importance.

The first parameter to tune is l, i.e. the amount of time spent in the search phase. A value of $l = 2$ means that twice as much time is spent in the local search phase than the construction phase. With γ fixed at 1, new experiments are conducted for different values of l. Each test case in the training set is solved once for each value of l, and with the execution time defined in Table 10. The benchmark results are shown in Table 12. In general, the benchmark results are very similar for all values of l, but $l = 2$ yields the lowest average queue usage. Thus, l is fixed at 2.

The same procedure is repeated for different values of π. Based on the results presented in Table 13, π is fixed at 4. In the same way, steepest ascent is chosen as the local search strategy based on Table 14. In conclusion, the following parameter values are chosen based on the tuning:

$$\gamma = 1, \quad l = 2, \quad \pi = 4, \quad a = \text{steepest ascent}$$

36.3 Compared to Optimum

Fig. 31 compares the metaheuristic with ILP-K for small topologies and hyperperiod 1 ms. On average, it uses 0.02 more queues than the optimum with

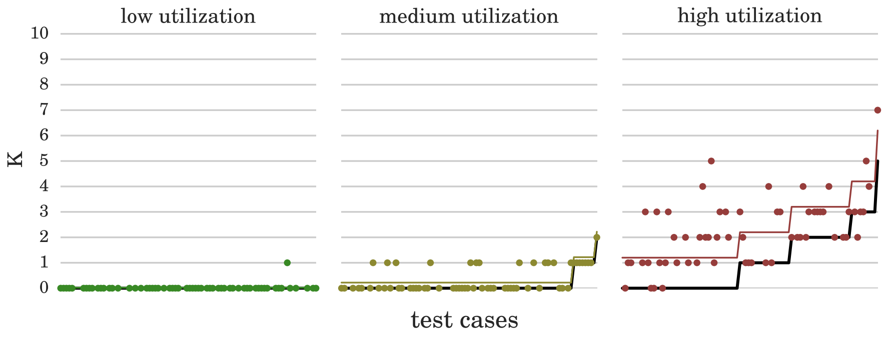


Figure 31: Metaheuristic queue usage compared to optimum for smallest topologies with hyperperiod 1 ms.

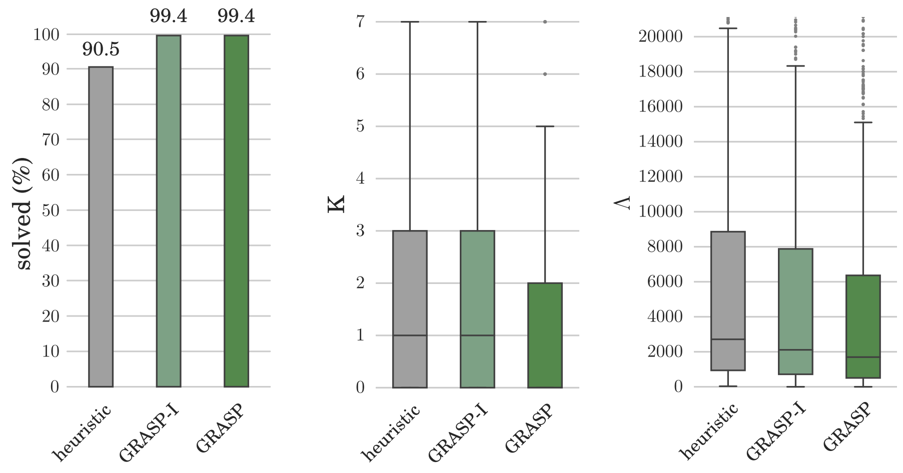
low utilization, 0.22 extra queues with medium utilization, and 1.2 extra queues with high utilization. It solves 98%, 78%, and 33% of test cases to optimality with low, medium, and high utilization, respectively. Comparing with Fig. 29, the metaheuristic is on average better than the BEST heuristic.

Notice, that there is one data point below the optimal value for high link utilization. This is due to the fact ILP-K does not finish within the time limit for this test case. The best feasible solution found by ILP-K is assumed optimal, which turns out to be a wrong assumption in this particular case, as the metaheuristic is able to find a better solution.

36.4 Compared to Heuristic

The GRASP method is compared to the BEST heuristic, which is defined as the best of the twelve heuristic variations for each test case. However, ASAP solves all test cases because it was used as a proxy-function to construct the test cases. This gives an unfair comparison. Thus, the average solve percentage among the twelve variations is used instead.

We introduce a third strategy, GRASP-I, which denotes a variation of GRASP that terminates once a complete, feasible solution has been found. Comparing GRASP-I with GRASP gives an indication of how much the solution quality improves over time.



(a) solve percentage **(b)** queue usage **(c)** end-to-end latency

Figure 32: Comparison of heuristic, GRASP-I, and GRASP.

Fig. 32a shows the solve percentage for the three strategies evaluated on the test set. As GRASP only improves upon the initial solution it is expected that GRASP and GRASP-I solve the same number of test cases.

Fig. 32b and 32c show queue usage and end-to-end latencies for the three strategies. GRASP is able to reduce queue usage and end-to-end latency significantly compared to both the heuristic and GRASP-I. The figure shows that GRASP is able to schedule 50% of test cases using no excess queues, and 75% are solved using no more than two excess queues. For end-to-end latency, 50% are solved with a total latency less than 2000, and 75% are solved with latency less than 7000. The figures also indicate that GRASP-I improves end-to-end latency slightly compared to the heuristic, but does not seem to improve queue usage.

Fig. 33 shows queue usage for different levels of link utilization. The box plots clearly show that GRASP outperforms the heuristic at high link utilization. On average, GRASP uses 2.9 excess queues at high utilization, where the heuristic uses 5.9 excess queues. Thus, a reduction of 51%. When considering end-to-end latency the improvement was 18%. See Fig. 36 and 37 in Sect. 39 for details.

GRASP-I uses 5.1 queues on average for high utilization, thus GRASP improves the queue usage by 43%. Furthermore, GRASP reduces end-to-end latency by

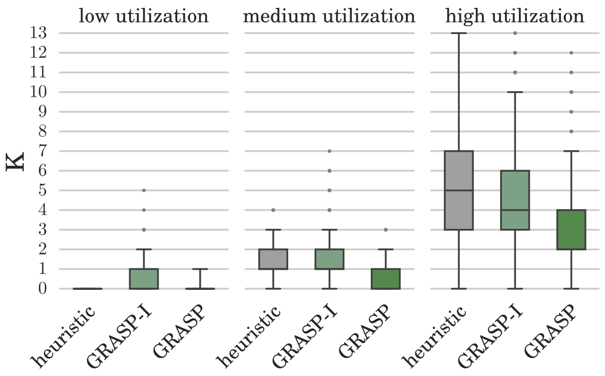


Figure 33: Queue usage of heuristic, GRASP-I, and GRASP at different link utilizations.

	Time	**Iterations**
Average	5.5s	1.1
Median	1.4s	1
Maximum	60.7s	6

Table 15: Running time and iteration count for GRASP-I. This corresponds to the time and iteration at which GRASP finds the first feasible solution.

2%. Table 15 show timing statistics for GRASP-I. At most six GRASP iterations are needed to find a feasible schedule in around one minute. In half of the cases only a single iteration and less than two seconds were sufficient to find the first feasible solution.

37 Conclusions and Future Work

This section presents the main conclusions of the report and discusses ideas for future research.

38 Conclusions

In this report we have proposed methods for configuring communication networks in safety-critical real-time systems. We presented the problem of scheduling Time-Triggered (TT) traffic in time-sensitive networks as defined in the *IEEE 802.1Qbv* standard. Periodic messages, called flows, are fragmented into frames which are sent from sender to receiver via network switches. Frames are forwarded in a time-triggered manner by configuring Gate-Control Lists (GCLs). The GCLs define when to open and close gates associated with each egress port queue in network switches.

The problem of configuring GCLs corresponds to scheduling the temporal offset of each frame within its period, and assigning flows to egress port queues. Schedules must satisfy several constraints in order to be feasible. In particular, we have identified a constraint needed to ensure deterministic queue forwarding: Flows cannot wait in a queue at the same time. As a result, frames of different flows must be temporally or spatially isolated, such that they either do not use the queue at the same time, or are assigned different queues.

The problem was defined as a multi-objective optimization problem, minimizing the total number of queues used by time-triggered flows, as well as their end-to-end latencies. We have showed that the problem is NP-complete, implying that no polynomial-time algorithm finds the optimal solution. Hence, any algorithm is a trade-off between tractability and solution quality. In this report we have proposed three strategies, each representing a different choice for this trade-off.

The first strategy is an Integer Linear Programming (ILP)-based approach. It minimizes an objective function, which is a combination of the two objective metrics. The objective function is minimized while satisfying a set of constraints defining the solution space of feasible schedules. There are many advantages to this approach: It finds proven optimal solutions, it is guaranteed to find a solution if one exists, and it is easy to add constraints to the model once they are formalized. It has one major drawback, though: Its running time is exponential in the worst case, making it intractable for industrial-sized applications.

The second strategy is a constructive heuristic strategy. It schedules flows one-by-one in a predetermined order, attempting to minimize queue usage. Once a flow has been successfully scheduled, the frame offsets are post-processed to reduce end-to-end latency. The primary advantage of this strategy is its polynomial running time. However, it lacks the guarantees of the ILP approach. It is not guaranteed to find a solution even if one exists, and the produced schedules could be far from optimal. Another drawback is the complexity of the implementation. The heuristic must be carefully implemented to make sure that the produced schedules are feasible, i.e., satisfy all constraints. Adding additional constraints or changing the objective function could mean redesigning the heuristic.

The third strategy is a GRASP-based metaheuristic, which generalizes the constructive heuristic into twelve unique heuristics. Each flow may be scheduled using any of the twelve heuristics. The metaheuristic performs a search space exploration to find the best heuristic for each flow. Every GRASP iteration consists of two phases: A construction phase, that constructs an initial solution in a greedy randomized manner, and a local search phase, which searches for a local optimum by destroying and repairing parts of the schedule. The motivation behind the metaheuristic is to give a tractable alternative to the ILP approach, while being more adaptable and provide better solutions than the heuristic approach. The metaheuristic relies on the heuristic for scheduling individual flows, thus it adopts its implementational drawbacks. In addition, the metaheuristic comes with several parameters which are experimentally tuned to explore the search space in the best way.

The three methods were experimentally evaluated on a set of synthetically generated industrial-sized test cases. The test cases were categorized by relevant parameters, such as the size of the network and the level of link utilization. Test cases with high link utilization are more difficult to schedule and will in general require more queues. For high link utilization the ILP approach struggled to find optimal solutions even for test cases with a few hundred frames. Half of the high utilization test cases timed out after four hours with the ILP approach.

For comparison, the heuristic approach required only seconds to solve the largest test cases with tens of thousands of frames. However, when considering high link utilization, the heuristic used 3.0 extra queues on average, and had an average solve percentage of 90%.

The metaheuristic approach showed significant improvements compared to the heuristic. It solved nearly all test cases (99.4%), and reduced queue usage as well as end-to-end latency. At high link utilization, it used 1.2 more queues than the optimum, compared to 3.0 of the heuristic. On average, the metaheuristic reduced queue usage by 51% and end-to-end latency by 18%, when considering high link utilization. The execution time of the metaheuristic was set between one minute and one hour, but feasible solutions were found within one minute for all test cases. The experimental evaluation shows that the GRASP metaheuristic is a promising method for finding high-quality schedules for large industrial-sized instances. The following section discusses directions for future work related to the metaheuristic.

39 Future Work

The metaheuristic approach can be extended and optimized in many ways to improve performance and add functionality. Some interesting directions for future work are:

- Relaxing the assumption that routes are given. It is straightforward and computationally inexpensive to extend the neighborhood of the metaheuristic to consider alternative routes. For instance, a set of alternative routes may be precomputed for each flow, and the metaheuristic could randomly choose one route when scheduling the flow. Implementing such functionality in an ILP model is much more involved and will add to the complexity of an already intractable problem.

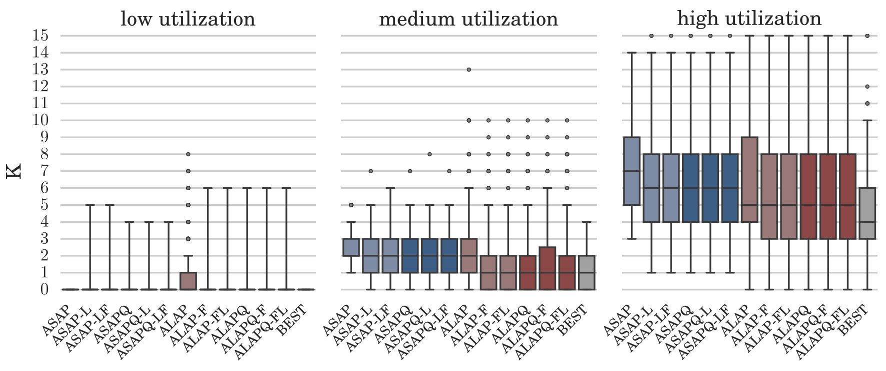
- Utilizing preemption between time-triggered queues, as defined in *IEEE 802.1Qbu*. In this report preemption is only considered in the context of preempting lower priority traffic to make way for time-triggered traffic. However, it is possible to let one TT flow preempt another, given that they are assigned different queues. This enables finding feasible schedules in cases where the current solution will fail. Preemption is particularly useful in scenarios where the feasible regions are too fragmented to transmit complete frames. Like alternative routes, such an extension is not easily implemented in an ILP model.

- Extending the metaheuristic neighborhood with additional scheduling methods. For instance, a modification of the presented ILP approach could be used to solve a small subproblem to optimality, thereby arriving at a local optimum. Consider that two flows are removed from the current schedule as part of the local search. An ILP model could then find the optimal way of rescheduling these two flows in the current schedule. If the two flows have a limited number of frames, the experimental evaluations indicate that the subproblem could be solved to optimality in seconds. This will allow the metaheuristic to explore parts of the search space that are unreachable with the current neighborhood definition.
- Implementing delta evaluation for the local search. The current implementation features an approach, where the entire schedule is copied for every iteration of the local search (see Sect. 31.2). Hence, the running time is proportional to the total number of frames in the schedule. With clever bookkeeping it is possible to delete flows in the current schedule without copying. This involves reverting the impact of each of the deleted flows, i.e., adding back intervals to the feasible region, predicting the new objective value, etc. Implementing delta evaluation will improve performance of the metaheuristic, which enables it to complete more iterations in the same amount of time. Consequently, it will find improvements more quickly. Especially large topologies, where flows cover only small parts of the network, will benefit from delta evaluation.
- Introducing configurable priorities for the multi-objective function. In the presented method, queue usage is always considered more important than end-to-end latency. Hence, end-to-end latency is only minimized if it is not possible to reduce the queue usage. It is straightforward to redefine the objective function like presented in Sect. 18, where constants c_1 and c_2 define the priorities of queue usage and latencies, respectively. This will allow the system engineer to set c_1 and c_2 to reflect the schedules of interest. However, the metaheuristic relies on the heuristic approach for scheduling individual flows, and the heuristic is designed to primarily reduce queue usage. Thus, in order to achieve good results the heuristic should be redesigned.

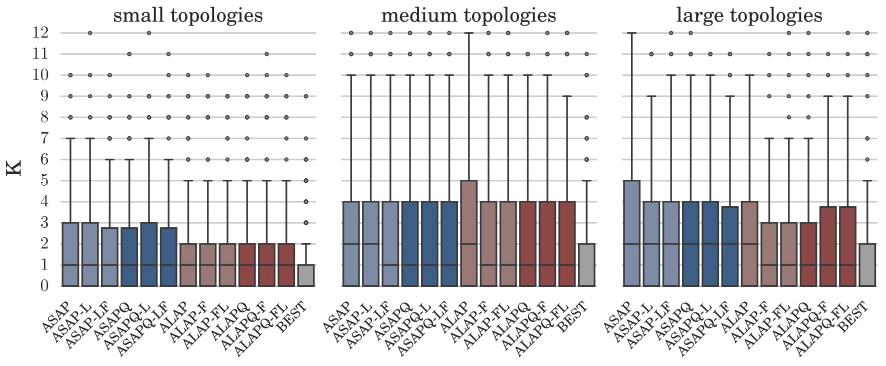
Additional Evaluations

The following pages contain box plots of queue usage and end-to-end latency for the three test case dimensions: Link utilization, topology size, and hyperperiod. Fig. 34 and 35 visualize the distributions for queue usage and end-to-end latency for the heuristic variations, and Fig. 36 and 37 visualize the same for the metaheuristic.

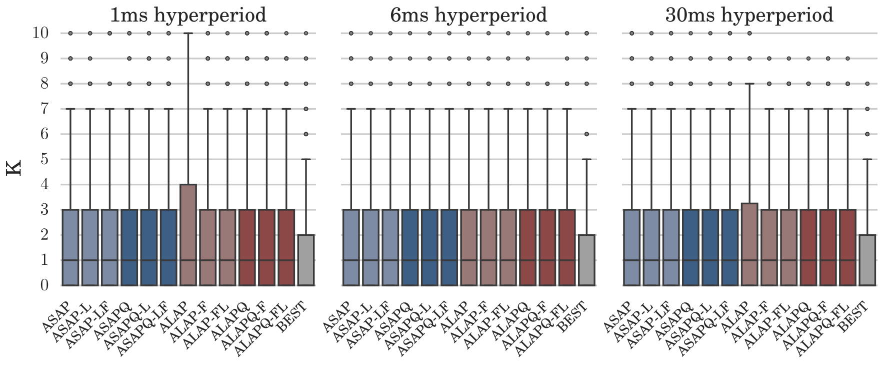
Note that Fig. 34 and 35 are based on all 1620 test cases, whereas Fig. 36 and 37 are only based on the test set (1080 test cases). The figures are meant as a supplement to Sect. 32.



(a) link utilization

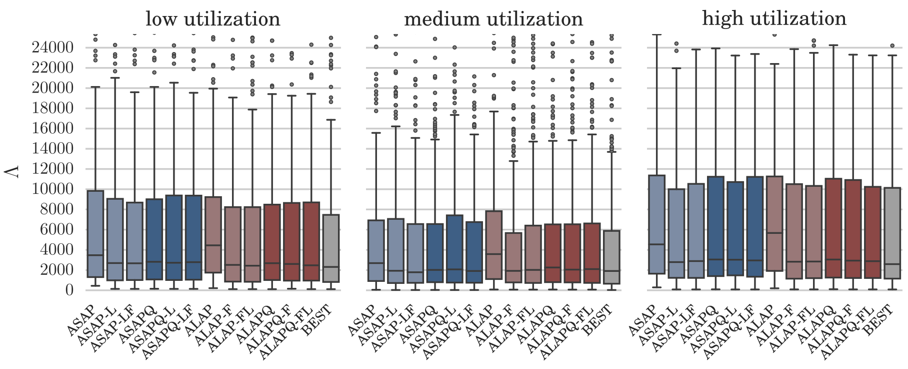


(b) topology size

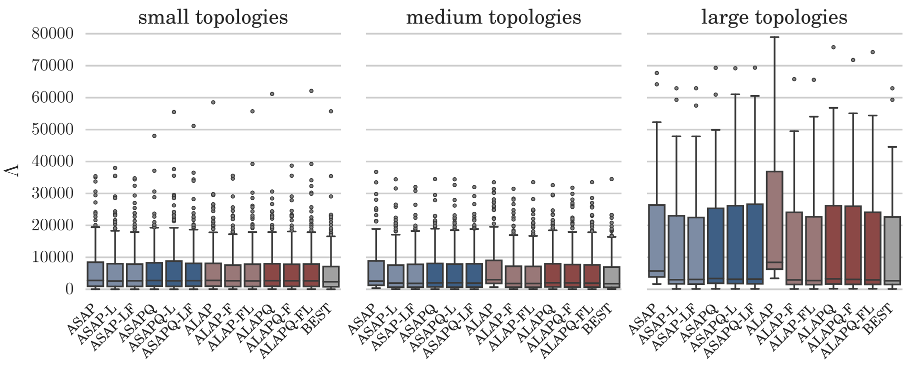


(c) hyperperiod

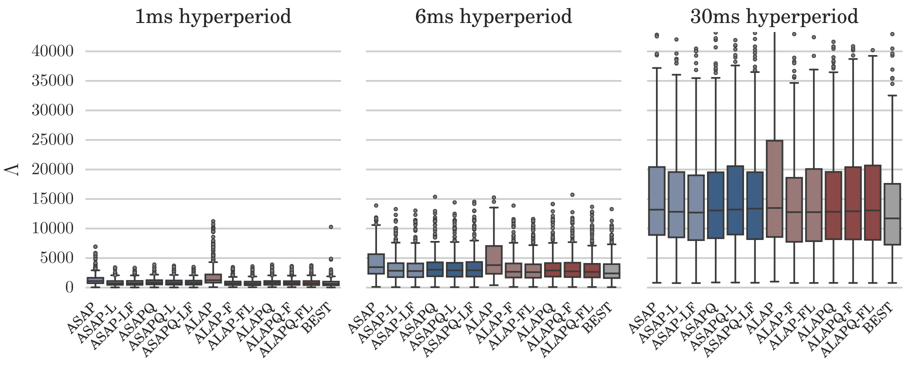
Figure 34: Queue usage comparison for twelve heuristic variations. The figures show boxplots for varying test case parameters.



(a) link utilization

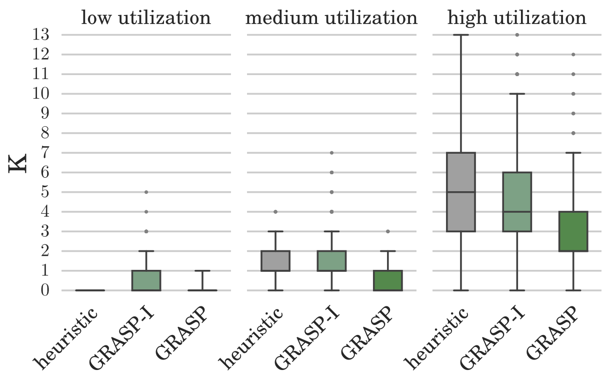


(b) topology size

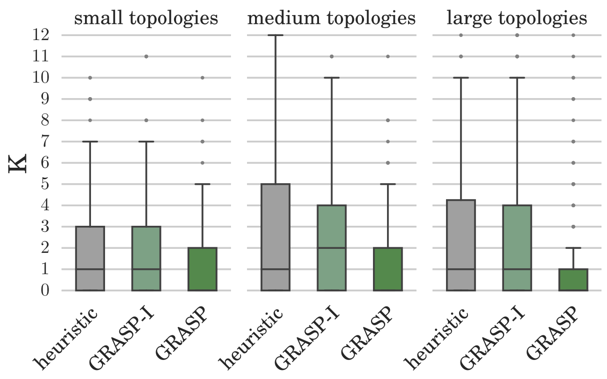


(c) hyperperiod

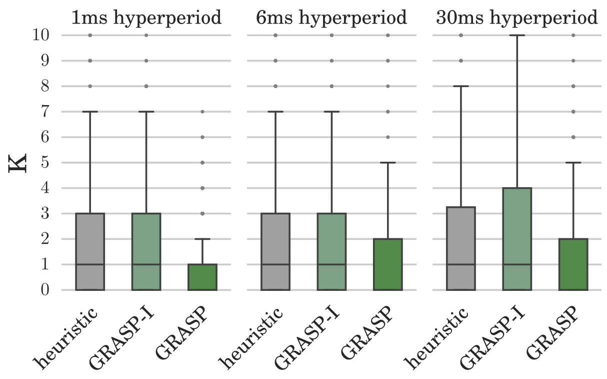
Figure 35: End-to-end latency comparison for twelve heuristic variations. The figures show boxplots for varying test case parameters.



(a) link utilization

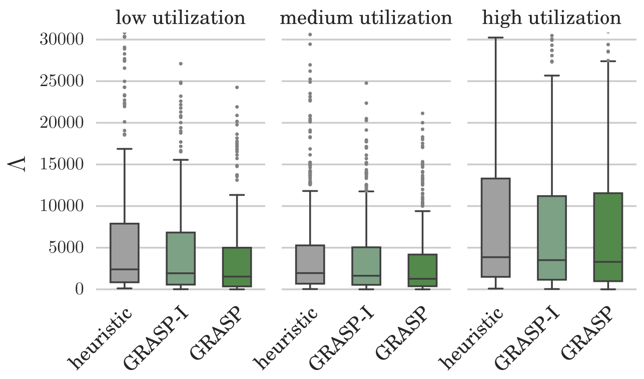


(b) topology size

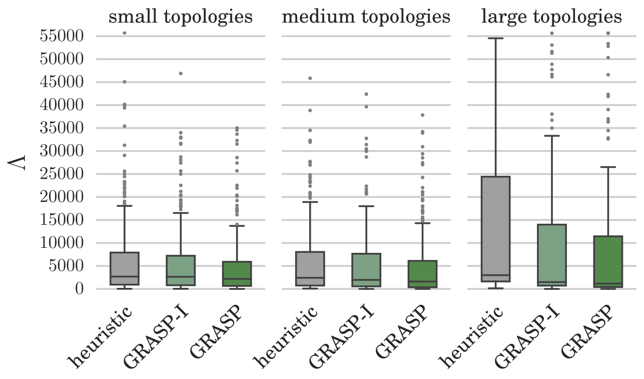


(c) hyperperiod

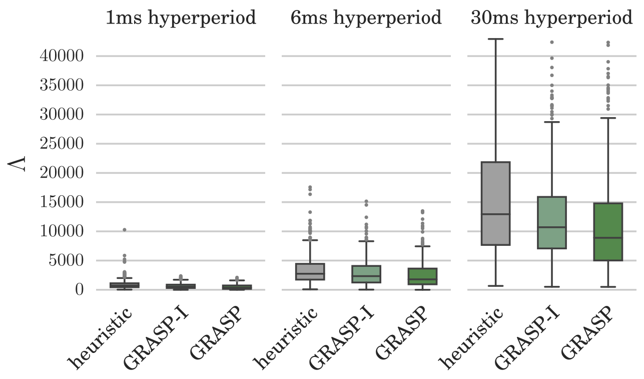
Figure 36: Queue usage comparison for best heuristic and GRASP. The figures show boxplots for varying test case parameters.



(a) link utilization



(b) topology size



(c) hyperperiod

Figure 37: End-to-end latency comparison for best heuristic and GRASP. The figures show boxplots for varying test case parameters.

References

[1] G. C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Norwell, MA, USA: Kluwer Academic Publishers, 1997.

[2] *IEEE TSN (Time-Sensitive Networking): A Deterministic Ethernet Standard*, TTTech Computertechnik AG, 2015.

[3] F. Dürr and N. G. Nayak, "No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN)," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. New York, NY, USA: ACM, 2016, pp. 203–212. [Online]. Available: http://doi.acm.org/10.1145/2997465.2997494

[4] A. Gilchrist, *Industry 4.0: the Industrial Internet of Things.* Apress, 2016.

[5] *ISO 11898: Road vehicles – Controller area network (CAN)*, International Organization for Standardization, 2003.

[6] *ISO 17458: Road vehicles - FlexRay communications system*, 1st ed., International Organization for Standardization, 2013.

[7] J. Specht and S. Samii, "Urgency-Based Scheduler for Time-Sensitive Switched Ethernet Networks," *Proc. of Euromicro Conference on Real-time Systems*, pp. 75–85, 2016.

[8] *802.3 Standard for Ethernet*, IEEE, 2015.

[9] J. D. Decotignie, "Ethernet-Based Real-Time and Industrial Communications," *Proc. of the IEEE*, vol. 93, no. 6, pp. 1102–1117, 2005.

[10] Z. Lin and S. Pearson, *An inside look at industrial Ethernet communication protocols*, Texas Instruments, 2013.

[11] *EtherCAT Leopard-like Speed and Efficiency*, Real Time Automation, 2008.

[12] R. Pigan and M. Metter, *Automating with PROFINET*, 2nd ed., Publicis Publishing.

[13] E. Schemm, "SERCOS to link with ethernet for its third generation," *Computing and Control Engineering*, vol. 15, no. 2, pp. 30–33, 2004.

[14] *AS6802: Time-triggered Ethernet*, SAE International, 2011.

[15] IEEE, "Official Website of the 802.1 Time-Sensitive Networking Task Group," http://www.ieee802.org/1/pages/tsn.html, 2016, [Online; accessed 30-November-2016].

[16] T. Steinbach, H. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Tomorrow's in-car interconnect? A competitive evaluation of IEEE 802.1 AVB and time-triggered ethernet (AS6802)," in *Proceedings of the 76th IEEE Vehicular Technology Conference, VTC Fall 2012, Quebec City, QC, Canada, September 3-6, 2012*, 2012, pp. 1–5. [Online]. Available: http://dx.doi.org/10.1109/VTCFall.2012.6398932

[17] T. Herpel, B. Kloiber, R. German, and S. Fey, "Routing of safety-relevant messages in automotive ecu networks," in *Vehicular Technology Conference Fall (VTC 2009-Fall), 2009 IEEE 70th.* IEEE, 2009, pp. 1–5.

[18] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: Distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. New York, NY, USA: ACM, 2013, pp. 69–84. [Online]. Available: http://doi.acm.org/10.1145/2517349.2522716

[19] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, 2016. [Online]. Available: http://dx.doi.org/10.1007/s11241-015-9244-x

[20] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task-and network-level schedule co-synthesis of ethernet-based time-triggered systems," in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC).* IEEE, 2014, pp. 119–124.

[21] W. Steiner, "An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks," in *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st.* IEEE, 2010, pp. 375–384.

[22] ——, "Synthesis of static communication schedules for mixed-criticality systems," in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on.* IEEE, 2011, pp. 11–18.

[23] F. Pozo, W. Steiner, G. Rodriguez-Navas, and H. Hansson, "A decomposition approach for smt-based schedule synthesis for time-triggered networks," in *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on.* IEEE, 2015, pp. 1–8.

[24] D. Tămaş-Selicean, P. Pop, and J. Madsen, "Design of mixed-criticality applications on distributed real-time systems," *Technical University of Denmark*, 2014.

[25] G. Avni, S. Guha, and G. Rodriguez-Navas, "Synthesizing time-triggered schedules for switched networks with faulty links," in *Embedded Software (EMSOFT), 2016 International Conference on.* IEEE, 2016, pp. 1–10.

[26] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. New York, NY, USA: ACM, 2016, pp. 183–192. [Online]. Available: http://doi.acm.org/10.1145/2997465.2997470

[27] P. Pop, M. L. Raagaard, S. S. Craciunas, and W. Steiner, "Design Optimization of Cyber-Physical Distributed Systems using IEEE time-sensitive Networks (TSN)," *IET Cyber-Physical Systems: Theory & Applications*, pp. to–appear, 2016.

[28] IEEE, "802.1AS-Rev - Timing and Synchronization for Time-Sensitive Applications," http://www.ieee802.org/1/pages/802.1AS-rev.html, 2016, [Online; accessed 30-November-2016].

[29] ——, "802.1Qav - Forwarding and Queuing Enhancements for Time-Sensitive Streams," http://www.ieee802.org/1/pages/802.1av.html, 2009, draft 7.0.

[30] ——, "802.1Qbv - Enhancements for Scheduled Traffic," http://www.ieee802.org/1/pages/802.1bv.html, 2016, [Online; accessed 30-November-2016].

[31] ——, "802.1Qbu - Frame Preemption," http://www.ieee802.org/1/pages/802.1bu.html, 2016, [Online; accessed 30-November-2016].

[32] *802.1Q Bridges and Bridged Networks*, IEEE, 2014.

[33] G. A. Ditzel and P. Didier, "Time Sensitive Network (TSN) Protocols and use in EtherNet/IP Systems," *2015 ODVA Industry Conference & 17th Annual Meeting*, 2015.

[34] O. Sinnen, *Task Scheduling for Parallel Systems*, ser. Wiley Series on Parallel and Distributed Computing. Wiley, 2007.

[35] T. H. Cormen, C. E., R. L., and C. Stein, *Introduction to Algorithms*, 2nd ed. MIT Press and McGraw–Hill, 2001, ch. Section 24.3: Dijkstra's algorithm, pp. 595—-601.

[36] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976. [Online]. Available: http://dx.doi.org/10.1287/moor.1.2.117

[37] "CPLEX Optimizer," https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/, [Online; accessed 17-December-2016].

[38] "Gurobi Optimizer," http://www.gurobi.com/products/gurobi-optimizer, [Online; accessed 17-December-2016].

[39] S. Venugopalan and O. Sinnen, "ILP Formulations for Optimal Task Scheduling with Communication Delays on Parallel Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 1, pp. 142–151, Jan 2015.

[40] "Cython, C-Extensions for Python," http://cython.org/, [Online; accessed 24-January-2017].

[41] F. Glover and M. Laguna, "Tabu search, 1997," *Kluwer Academic Publishers*, 1997.

[42] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.

[43] M. G. Resende and C. C. Ribeiro, "Grasp: greedy randomized adaptive search procedures," in *Search methodologies.* Springer, 2014, pp. 287–312.

[44] P. Hansen and N. Mladenovic, "Variable neighborhood search," in *Search methodologies.* Springer, 2014, pp. 313–338.

[45] R. S. Oliver, S. S. Craciunas, and G. Stöger, "Analysis of deterministic ethernet scheduling for the industrial internet of things," in *2014 IEEE 19th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD).* IEEE, 2014, pp. 320–324.

[46] T. W. Kirkman, "Statistics to Use," http://www.physics.csbsju.edu/stats/, 1996, [Online; accessed 29-January-2017].