No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN)

Frank Dürr, Naresh Ganesh Nayak
Institute for Parallel and Distributed Systems, University of Stuttgart
{duerrfk, nayaknh}@ipvs.uni-stuttgart.de

ABSTRACT

The IEEE Time-sensitive Networking (TSN) Task Group has recently standardized enhancements for IEEE 802.3 networks for enabling it to transport time-triggered traffic (*aka* scheduled traffic) providing them with stringent bounds on network delay and jitter while also transporting best-effort traffic. These enhancements primarily include dedicating one queue per port of the switch for scheduled traffic along with a programmable gating mechanism that dictates which of the queues are to be considered for transmission. While the *IEEE 802.1Qbv* standards define these mechanisms to handle scheduled traffic, it stops short of specifying algorithms to compute fine-grained link schedules for the streams of scheduled traffic. Further, the mechanisms in TSN require creation of so-called *guard bands* to isolate scheduled traffic from the best-effort traffic. These guard bands may potentially result in bandwidth wastage, and hence schedules with lower number of guard bands are preferred. In this paper, we introduce the No-wait Packet Scheduling Problem (NW-PSP) for modelling the scheduling in IEEE Time-sensitive Networks and map it to the No-wait Job-shop Scheduling Problem (NW-JSP), a well-known problem from the field of operational research. In particular, we present a Tabu search algorithm for efficient computing of schedules and a schedule compression technique to reduce number of guard bands in schedule. Our evaluations show that our Tabu search algorithm can compute near-optimal schedules for over 1500 flows and the subsequent schedule compression reduces the number of guard bands on an average by 24 %.

CCS Concepts

•**Networks** → *Network resources allocation; Network design and planning algorithms;*

Keywords

Time-sensitive network, TSN, Real-time communication, Job shop scheduling problem, Tabu search, IEEE 802.1Qbv

RTNS '16 Brest, France

DOI: 10.475/123_4

1. INTRODUCTION

One central property of the popular Industry 4.0 paradigm is networked cyber-physical systems, where physical processes are controlled by computers. Sensors capture the state of the plant, transmit sensor values to controllers, which in turn send commands to actuators to influence the state of the plant and keep it close to the desired setpoint. Since sensor and actuators are possibly distributed accross the plant, a communication network is required to connect sensors, actuators, and controllers.

Since many physical processes such as the motion control of a set of cooperating robots are highly time-sensitive, real-time communication networks are required to control these systems. In order to guarantee a deterministic behavior of the physical system under control, a real-time network with deterministically bounded network delay and delay variation (jitter) is required. Traditionally, field buses have been used for this purpose. Later, with the great success of the Ethernet technology, different real-time Ethernet technologies have been proposed such as SERCOS III [16], PROFINET [20], etc.

Although these Ethernet extensions provide deterministic real-time properties with cycle-times down to 31.5 μs and the possibility to transmit both, real-time and non-real-time traffic, over the same medium, they are incompatible with each other. Hence, different technologies cannot be operated on the same physical medium without losing real-time guarantees.

Realizing the need for real-time Ethernet technologies in many application domains and the problem of incompatibilities within existing real-time Ethernet extensions, the Institute of Electrical and Electronics Engineers (IEEE) has started to standardize enhancements for deterministic real-time IEEE 802 networks. In particular, the *IEEE 802.1Qbv* [4] defines enhancements for the so-called scheduled traffic. The basic concept is to utilize the precise clock synchronisation of switches and end systems (up to nanosecond precision) as achieved by the IEEE 1588 (Precision Time Protocol (PTP)) [3] or IEEE 802.1AS [2] to schedule the forwarding of frames in the switched network. Frames are injected into the network by the source network interface controller (NIC) at precisely defined points in time according to a schedule. Moreover, switches along the path schedule access to the medium for scheduled and best-effort traffic using their synchronized clocks such that packets belonging to scheduled traffic are guaranteed timely access to the medium. When exactly queues for scheduled traffic and best-effort traffic get access to the medium is determined by

the so-called *gates*. The gates of a switch are regulated by a *gate driver program* which precisely defines when they are opened or closed.

Although the *IEEE 802.1Qbv* defines the basic scheduling mechanisms (time-triggered gates), how to configure the schedules to achieve bounded end-to-end network delay is beyond the scope of the standard. Therefore, we consider the problem of calculating optimized TSN schedules for NICs and switch gate drivers in this paper. Specifically, we consider the *offline calculation* of schedules for periodic *time-triggered traffic*. To this end, we show how the well-known *No-wait Job-shop Scheduling Problem* [14] can be adapted to a *No-wait Packet Scheduling Problem* (NW-PSP) for calculating TSN schedules yielding minimum network delay for real-time flows and compact schedules as a first contribution. We show that instances of NW-PSP can be formulated as Integer Linear Programs (ILP) for calculating exact solutions. However, since NW-PSP is NP-hard, ILPs are not scalable beyond very small problem instances. Therefore, as a second contribution, we propose a *heuristic optimization algorithm* based on the Tabu search metaheuristic that allows for efficient schedule calculation. As a third contribution, we show how to further optimize calculated schedules through *schedule compression* to further reduce the number of gate-driver entries and to reduce wasted bandwidth due to *guard-bands* between scheduled and best-effort traffic. Thus, overall, the calculated schedules guarantee minimum network delay for time-sensitive flows, are compact, and can be calculated efficiently. Our evaluations show that using this heuristic near-optimal schedules can be calculated for large networks with over 1500 time-sensitive flows in less than three hours with an average reduction of 24 % in the number of gate-driver entries.

The remainder of this paper is structured as follows. We introduce our system model and problem statement in Sec. 2 and 3, respectively. In Sec. 4, we describe the NW-PSP based on No-wait Job-shop Scheduling and its ILP formulation. In Sec. 5, we present the efficient Tabu search algorithm for NW-PSP along with our approach to compress schedules, before presenting the evaluation results in Sec. 6. Finally, we discuss related work in Sec. 7 and conclude the paper in Sec. 8.

2. SYSTEM MODEL

Time-sensitive networks comprise network elements (switches) compliant with the *IEEE 802.1Qbv* standards along with a network controller for programming them, for instance, using the Simple Network Manangement Protocol (SNMP). As shown in Fig. 1, these TSN switches are designed with additional enhancements for carrying scheduled traffic along with best-effort traffic [4]. The switches dedicate one predefined queue (say queue corresponding to traffic class 7) per port to handle scheduled traffic. Each queue in a TSN switch is equipped with a gate regulated by a programmable gate-driver. Packets in a queue are considered for transmission only if the corresponding gate is in "Open" state. Typically, the gate-driver program is a sequence of two-tuple entries representing gate events, where the first part of each entry is a timestamp relative to the start of the program, while the second part is a bitmask that determines the opened gates subsequent to lapse of the mentioned time. For instance, in Fig. 1, after time T_2 all gates are closed, while after time T_3 only the

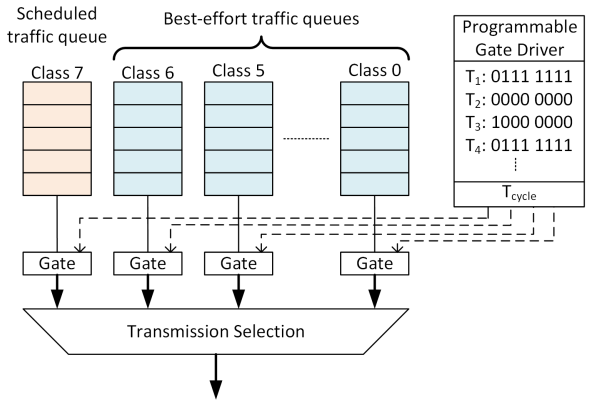


Figure 1: Architecture of a single port in an TSN switch (compliant with IEEE 802.1Qbv) [12].

gate for scheduled traffic is open. Additionally, after a pre-programmed time, T_{cycle}, the program terminates and restarts, thus providing a cyclic schedule of length T_{cycle} for regulating gates. Typically, all the gate drivers in a given network are programmed with the same value of T_{cycle}.

The network controller computes the gate-driver schedules by exploiting its knowledge of the network topology (gathered using Link Layer Discovery Protocol (LLDP)) along with the specifications of scheduled traffic in the network. Each time-sensitive flow (stream of scheduled traffic) is specified by its source host, destination host, and the amount of data sent each cycle. In this paper, we restrict flows to have cycle times equalling T_{cycle}. The network controller programs the gate-drivers in the switches based on the computed schedules, and also provides the source hosts of the time-sensitive flows with a timestamp relative to the start of the program for injecting packets of scheduling traffic into the network. The source host of the time-sensitive flows adheres with the assigned schedule and tags the packets of scheduled traffic with a IEEE 802.1Q header with the Priority Code Point (PCP) field corresponding to the queue dedicated for scheduled traffic (7 as per Fig. 1).

Further, the clocks of all TSN switches and hosts in the network are synchronised using IEEE 802.1AS or IEEE 1588 protocol which provides sub-microsecond precision. This also enables the TSN switches to synchronize the start of cycles, and to atomically update schedules across the network, in case of incremental future updates. It must also be noted that as IEEE Time-sensitive Networks are targeted towards local area networks, we assume for the scheduling problem that the diameter of the network is bounded to 8 hops, in accordance to the IEEE 802.1D standard.

3. PROBLEM STATEMENT

The computation of fine-grained link schedules for scheduled traffic in time-sensitive networks, in general, is equivalent to a bin-packing and is thus NP-Hard [17]. Given a set of time-sensitive flows along with their routes, schedules resulting in minimal network delays for packets belonging to time-sensitive flows are preferred. However, other important considerations also exist.

Typically, the TSN switches isolate scheduled traffic from best-effort traffic by means of so-called "guard bands" using the gating mechanism. The width of guard bands, t_g,

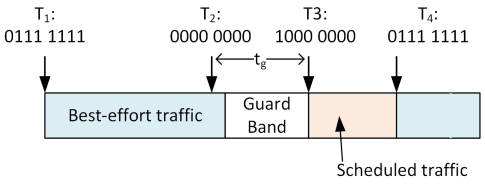


Figure 2: Port output on using Approach 1 to computing gate schedules in TSN

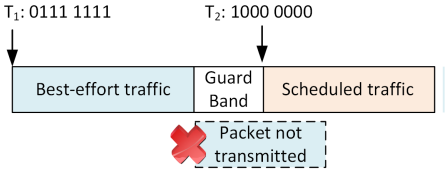


Figure 3: Port output on using Approach 2 to computing gate schedules in TSN

is the time required for serializing an MTU-sized packet on the port of the switch. The guard bands are created by closing the gates for best-effort traffic time t_g in advance before opening the gate for scheduled traffic. For instance, in Fig. 2, the gates for best-effort traffic are closed at time T_2, though the gate for scheduled traffic opens only at time T_3. This approach requires explicit computation of guard bands by the network controller during schedule generation to appropriately program the gate-drivers. Guard bands potentially lead to wastage of bandwidth for best-effort traffic. An alternative approach specified in the IEEE standards for TSN switches is to transmit a best-effort packet only if its transmission would finish before the corresponding gate closes and the gate for scheduled traffic opens, as shown in Fig. 3. While this approach does not need explicit computation of guard bands, they are implicitly created, as shown in Fig. 3. Thus, either ways, bandwidth wastage due to presence of guard bands is unavoidable.

Guard bands stem from the events in which the gate for scheduled traffic is opened. Increased number of such events in a schedule impacts the available bandwidth for best-effort traffic, and it is hence imperative to reduce the number of these events. This also results in compaction of gate driver programs. Note that these events cannot be completely eliminated, they can only be reduced by means of scheduling such that each gate opening event leads to transmission of several packets belonging to the scheduled traffic. To this end, we compute TSN schedules with minimal duration (also known as make-span) for time-sensitive networks, as reduced makespan also imply fewer gate opening events. Further, we compress the schedules using a procedure designed to explicitly reduce the number of gate opening events.

4. NO-WAIT PACKET SCHEDULING FOR TSN

The basic idea of this paper is to map the problem of calculating TSN transmission schedules to the well-known No-wait Job Shop Scheduling Problem. In this section we show how the original No-wait Job Shop Scheduling Problem has to be adapted to model packet scheduling in TSN by the corresponding No-wait Packet Scheduling Problem, and how this problem can be expressed as Integer Linear Program.

4.1 Background: Job Shop Scheduling

To define the TSN packet scheduling problem, we first briefly introduce the original *Job Shop Scheduling Problem (JSP)*. JSP is a well known scheduling problem from operational research. Informally, JSP can be described as follows. Given is a set of machines and set of jobs, each consisting of a sequence of operations like milling, drilling, etc. that have to be executed on the machines in the given sequence. Each operation can be executed on exactly one machine of the set of machines, e.g., a drilling operation on a drilling machine, and operations take a defined time duration. Moreover, each machine can only process one operation at a time. The JSP is a constrained optimization problem that tries to schedule each operation on the corresponding machine such that no more than one operation is processed at the same time on any machine (constraint). The schedule is expressed by the starting time of each operation of each job. Moreover, JSP tries to minimize the so-called *makespan*[1] of processing, where makespan is defined as the maximum of finishing times of the last operations for all jobs, i.e., we try to finish the set of jobs as fast as possible (objective).

The *No-wait Job Shop Scheduling Problem (NW-JSP)* adds an additional constraint: after a job is started, it cannot be interrupted, i.e., it must run to completion without time gaps between the processing of the operations of the job. With this constraint, the schedule can be defined by the starting time of each job alone. The no-wait property is important in many manufacturing scenarios. For instance, an iron has to be forged immediately after heating it up, and as we see later also has a meaning for packet scheduling.

It is worth to note that JSP is NP-complete and known as one of the most difficult combinatorial optimization problems [6], and NW-JSP is also NP-hard [13].

4.2 The No-wait Packet Scheduling Problem (NW-PSP) for TSN

We can now investigate how to adapt NW-JSP to calculate packet schedules for TSN. We call the corresponding problem the *No-wait Packet Scheduling Problem (NW-PSP)*. The overall goal is to define the times when packets are injected into the network by NICs at the source, and the times to open and close the gates for scheduled packets at switches.

The basic idea of mapping NW-JSP to NW-PSP is to map switches to machines performing forwarding operations on packets. A time-sensitive flow then corresponds to a sequence of forwarding operations, one for each switch along the given path of the flow. Packets should be forwarded immediately without delay, which intuitively corresponds to the no-wait property of NW-JSP.

In order to come up with a formal formulation of NW-PSP, we need to refine this basic idea further. In particular, the mapping "one switch, one machine" is too coarse-grained since typically a switch can forward several packets in parallel. So we need to ask the question: Which operations can (not) be performed in parallel during forwarding?

Moreover, network delay is more complex than the processing time of NW-JSP job operations. Network delay can

[1]Other objectives can be defined such as minimum tardiness, which are not useful for the packet scheduling problem.

be broken down into several types of delays, namely the propagation delay of signals along the link, the processing delay for deciding on which port to forward an incoming packet, the queueing delay of a packet in the queue of an outgoing port, and the transmission delay to serialize the packet on the wire. Note that the time intervals for propagation, processing, queuing, and transmission of each packet are ordered strictly sequentially and do not overlap since we assume a store-and-forward behavior for switches rather than cut-through forwarding. Before a packet can be processed, it must be received completely and put into an internal buffer. Packet processing then inspects the header fields and decides into which outgoing queue should the packet be enqueued. When processing is finished, the packet can be transmitted on the outgoing link as soon as it is at the head of the queue.

Due to physical restrictions, two packets cannot be transmitted over the same outgoing port at the same time since otherwise their electrical signals would interfere. Therefore, the transmission intervals of two packets using the same outgoing port must not overlap as shown for Flow 2 and Flow 3, both forwarded over port 5 of the same switch, in Figure 4. However, a switch can process several packets received in parallel from different incoming ports at the same time as shown for Flow 1 and Flow 2 in Figure 4. On similar lines, a network interface controller (NIC) cannot transmit two packets at the same time.

Based on this observation, *switch ports* and *NICs* now correspond to machines of NW-JSP. The set of all switch ports and NICs in the network is denoted as $P = \{P_1, \ldots, P_m\}$. *Time-sensitive flows* correspond to jobs, and we denote the set of flows as $F = \{F_1, \ldots, F_n\}$. Each flow F_i consists of a sequence of *transmission operations* $O_i = (O_{i,1}, \ldots, O_{i,n_i})$, with one transmission operation for each outgoing port along the given path of the flow. Relation $\mathcal{R} : O \mapsto P$ maps each transmission operation to an outgoing switch port, i.e., relation $\mathcal{R}$ corresponds the mapping of job operations to machines in NW-JSP. Sequence O_i together with relation $\mathcal{R}$ are parameters of NW-PSP defining the route of a flow through the network.

Similar to machines processing operations of a job, switch ports perform the transmission of packets of a flow. Note that according to this model, an operation only includes the transmission. Packet processing and propagation is not part of the operation! The processing time of a job then corresponds to the transmission delay of a packet, which is defined by the given packet size and data rate of the switch. Parameter $d_{i,j}^{trans}$ defines the transmission delay for each transmission operation $O_{i,j}$ of flow F_i.

In addition to the transmission delay, we also have to consider propagation and processing delays. We assume the same processing delay d^{proc} and propagation delay d^{prop} for all switches and links in the network. This simplifying assumption could be easily relaxed by defining individual processing and propagation delays based on individual switch and link properties, respectively. Due to the no-wait property, the queueing delay is per-definition zero.

In constrast to adjacent operations of a NW-JSP job, adjacent transmission operations on neighboring switches of a flow cannot be processed "back-to-back" without time gap. After one switch port has completed the transmission of the last bit of the packet, this bit first has to propagate to the neighboring switch, and then the packet needs to be processed before the transmission on the next outgoing port can start. Therefore, we need to consider the propagation and processing delays preceding each transmission interval. Let $D_{i,j}$ define the cumulative network delay (processing, propagation, and transmission delay) up to and including transmission operation $O_{i,j}$ of flow F_i:

$$D_{i,j} = (j-1)(d^{prop} + d^{proc}) + \sum_{k=1,\ldots,j} d_{i,k}^{trans} \qquad (1)$$

Then the essential constraint that two conflicting transmission operations $O_{i,k}$ and $O_{j,l}$ using the same outgoing port must not overlap in time can be defined as follows:

$$t_j - t_i \geq D_{i,k} - (D_{j,l-1} + d^{prop} + d^{proc}) \quad \text{or} \qquad (2)$$
$$t_i - t_j \geq D_{j,l} - (D_{i,k-1} + d^{prop} + d^{proc}) \qquad (3)$$

Here, t_i and t_j denote the start of the packet transmission on the source hosts for flows F_i and F_j, respectively. Informally, these constraints ensure that before the first bit of a packet from flow F_i is transmitted over a switch port, the last bit of the packet from flow F_j must have been transmitted over this port, or vice versa. $D_{j,l-1} + d^{prop} + d^{proc}$ defines the time when the packet is enqueued in the queue of the outgoing port after having transmitted this packet over the previous port, propagating the packet to the next switch, and processing the packet there. Note that packets are transmitted immediately after processing without waiting in the outgoing queue of the port (no-wait forwarding property of NW-PSP). Thus, packets travel through the network "non-stop" with *minimum possible network delay*! Another inherent advantage of no-wait forwarding is that the *queue size of real-time flows is minimal*, and switches can dedicate this memory to best-effort traffic.

The set of all flow start times $T = \{t_1, \ldots, t_n\}$, i.e., the time of injecting packet at source NIC, are the variables of NW-PSP. Given the start times, we can calculate packet schedules for transmitting packets on switches (gate opening times). A transmission operation $O_{i,k}$ needs to be scheduled at time $t_{i,k} = t_i + D_{i,j} - d_i^{trans}$ on port $\mathcal{R}(O_{i,k})$. This schedule is repeated in each cycle, i.e., $t = 0\,\text{s}$ defines the start of a cycle.

Note that simply repeating the schedule in each cycle only works because we assumed that all flows have the same cycle period. If flows have different cycle periods, this might lead to "collisions" A collision does not necessarily mean that the packet is dropped, however, they are queued since another packet is still in transmission over the same outgoing port. Obviously, this violates our goal of no-wait packet scheduling. This problem can be solved by calculating schedules for a *hyper-cycle* of length $LCM(p_1, \ldots, p_n)$, where LCM denotes the least common multiple of all flow cycle periods p_i. For space restrictions, we do not further describe this common principle known from scheduling here.

The objective of NW-PSP is to *minimize the flowspan* C_{max}, which we define as the equivalent to the NW-JSP makespan:

DEFINITION 1. *Let* $C_i = t_i + (n_i + 1)(d^{prop} + d^{proc}) + \sum_{k=1,\ldots,n_i} d_{i,k}^{trans}$ *be the finishing time of flow* F_i. *Then the* flowspan $C_{max} = \max\{C_i | i \in \{1, \ldots, n\}\}$ *is the finishing time of the flow finishing last.*

Minimizing the flowspan results in compact schedules

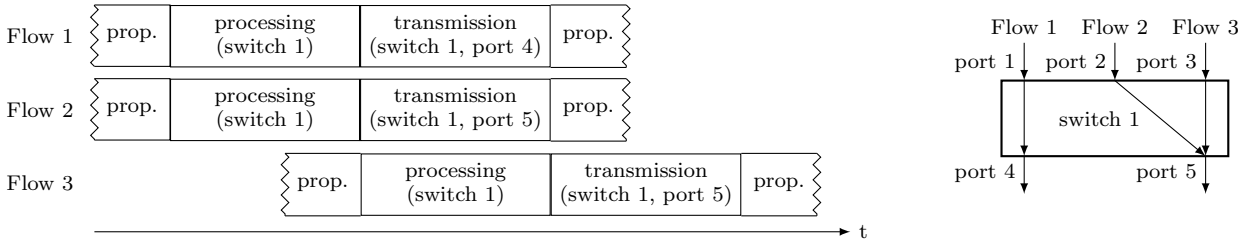


Figure 4: Timeline of forwarding three packets of three different flows over one switch. No queueing delay is shown since packets are forwarded immediately with no-wait packet scheduling.

where real-time flows are not distributed across the whole cycle, but are pressed to the start of the cycle. On the one hand, this increases the chance that packets from different real-time flows are scheduled back-to-back, such that the number of entries of gate-drivers can be minimized by merging gate opening times as shown later in Sec. 5.3. This results in reduced bandwidth wastage on account of guard bands. On the other hand, this gives best-effort flows larger continuous space within a cycle, where best-effort packets can be transmitted without interruption.

4.3 Integer Linear Program for NW-PSP

Similar to the orginal NW-JSP, also NW-PSP can be formulated as Integer Linear Program (ILP):

$$\min C_{max} \tag{4}$$

$$\text{subject to}$$

$$\forall \{O_{i,k}, O_{j,l}\} \in K:$$

$$t_j - t_i - D_{i,k} + D_{j,l-1} + d^{prop} + d^{proc} \leq c\, x_{i,k,j,l} \tag{5}$$

$$\forall \{O_{i,k}, O_{j,l}\} \in K:$$

$$t_i - t_j - D_{j,l} + D_{i,k-1} + d^{prop} + d^{proc} \leq c\,(1 - x_{i,k,j,l}) \tag{6}$$

The constraints of Equation 5 and 6 correspond to the constraints of Equation 2 and 3, respectively, after translating the disjunctive form to a conjunctive form as required by ILPs. To this end, we introduce binary variables $x_{i,k,j,l} \in \{0,1\}$ for each pair of conflicting forwarding operations $\{O_{i,k}O_{j,l}\}$ with the same outgoing port. Set $K = \{\{O_{i,k}O_{j,l}\} \,|\, \mathcal{R}(O_{i,k}) = \mathcal{R}(O_{j,l}) \wedge O_{i,k} \neq O_{j,l}\}$ defines all such pairs of conflicting forwarding operations. c is a large constant (virtually infinity). Depending on the value of x, either the first or second constraint is effective with a right-hand side value of zero. The ineffective constraint is then evaluating to true since "infinity" is greater than anything, ensuring the correct semantic of the conjunctive form.

5. HEURISTICS FOR NW-PSP

As already mentioned, the NW-PSP is an NP-hard problem. Therefore, as also our evaluations later show, we cannot expect to find exact solutions efficiently for larger scenarios with many flows using the ILP formulation from the previous section (these exact solutions can still serve as a reference). In order to improve scalability, we next present heuristics for efficiently solving the NW-PSP based on the Tabu search algorithm for NW-JSP in [13]. For an easier description, we make a few changes to the mapping between NW-PSP and NW-JSP. We now model switch ports by two machines—one for processing of incoming packets on the port (to model the processing delays and the propagation delays incurred by the packet), and one for transmitting the packets going out on the port as explained in Sec 4.2, i.e., we now use additional machines to account for processing and propagation delays of packets. Thus, a flow, now, consists of a sequence of operations (transmission and processing) executed by different machines. After a machine transmits a packet on a link, another machine responsible for the port on the next switch in its path processes it. While this mapping makes it easier to apply Tabu search for flowspan optimization of NW-PSP, this model also has the inherent shortcoming that processing of incoming packets cannot be achieved in parallel as in the unmodified problem formulation. However, such situations would not arise in most switches where transmission delays dominate processing delays.

With the no-wait constraints also applicable for NW-PSP, the overall schedule can be specified by the start times for each of the flows. The schedule for all the constituent operations of a flow can be calculated from its start time. The resulting schedule is then subjected to a specifically designed procedure to reduce the number of gate-opening events for scheduled traffic, thus reducing the number of guard bands and conserving bandwidth for best-effort traffic.

Based on the approach in [13], we split the NW-PSP into a *time-tabling* problem and a *sequencing* problem. The time-tabling problem deals with computation of start times for all flows belonging to a *totally ordered set of flows*. The sequencing problem, on the other hand, deals with *totally ordering the set of flows being scheduled* such that the given time-tabling algorithm results in a schedule with minimal flowspan. In the following, we describe the used time-tabling algorithm (based on a greedy approach) and the sequencing algorithm (based on Tabu search).

5.1 Time-tabling Problem

For timetabling, we use a greedy approach to compute start-times for the flows. Consider a totally ordered set of flows, $F \equiv \{F_1, F_2, \ldots, F_n | F_i \rightarrow F_j; \forall i \leq j\}$. The time-tabling algorithm, presented in Algorithm 1, allocates the earliest possible starting time (Line 4) for each flow in set F based on their order, one at a time, subject to the constraints imposed by the starting times of the preceding flows, i.e., no machine can process or transmit packets belonging to more than one flow at the same time. The algorithm sets initial start time of the flow being scheduled as 0, and increases it in steps (size based on set of conflicting operations of preceding flows) till all the operations of the current flow are scheduled without any conflicts with the preceding flows. Based on the start

times of the flows and its cumulative times for executing its operations, the flowspan is computed (Line 6).

While earliest possible starting time of flows as a heuristic do not always yield optimal solutions, they approximate schedules with minimal flowspan [13]. It may be noted that the worst-case time complexity of this time-tabling algorithm is $\mathcal{O}(n^3N^3)$, where n is the number of flows and N is the maximum number of constituent operations (transmission as well as processing) in any flows. For an NW-PSP instance corresponding to a local area network, the maximum number of constituent operations is bounded. This is because the length of path over a flow may be routed is bounded. Thus, the worst-case time complexity of the used time-tabling algorithm is $\mathcal{O}(n^3)$.

Algorithm 1 Time-tabling Algorithm

```
1: function TIMETABLER(SetOfFlows(F))
2:     Schedule ← { }, Span ← 0
3:     for each flow in F do
4:         FTime ← Earliest possible start time
5:         Schedule[flow] ← StartTime
6:         Span ← max(Span, FTime + flow.totalTime)
7:     return (Span, Schedule)
```

An important property of this presented time-tabling algorithm is that the starting time of flows is influenced only by the flows preceding it in the totally ordered set. The ones succeeding it play no role in determining its starting time. We can exploit this property for incremental expansion of schedules. For this, the flows yet to be scheduled can be appended to the current ordering of flows and the time-tabling algorithm can be re-executed. While the new sequence may not be an optimal one (in terms of the schedule flowspan), the schedules for existing flows will never be altered, an indispensable guarantee required to support incremental flow scheduling in time-sensitive networks.

5.2 Sequencing Problem

The sequencing algorithm creates a total ordering of flows in the set of flows, F, to minimize the resulting makespan of the schedule computed by the time-tabling algorithm. The search space for the sequencing problem consists of $n!$ possibilities, where n is the number of flows in the NW-PSP instance. A brute force approach to compute the optimal solution would involve each of the possible sequences to be executed with the presented time-tabling algorithm (complexity of $\mathcal{O}(n^3)$), and would therefore not scale to large problem sizes. Hence, we use Tabu search for a guided exploration of the solution space.

Tabu search is a well defined method for exploration of solution space in optimization problems [10]. Tabu search mainly generates an initial solution (based on a heuristic) and iteratively processes it by selecting the best possible solution in the neighbourhood that does not violate certain criterion (being on the tabu-list) for the next iteration. After a pre-defined number of iterations, the best-ever solution encountered over these iterations is selected. We adapt the Tabu search developed for solving a NW-JSP (from [13]) to solve an instance of NW-PSP modelling the scheduling problem in time-sensitive networks. The existing algorithms for NW-JSP have been primarily designed and evaluated for problem with 30–50 jobs. For NW-PSP, we aim for a solution that scales up to 1000+ flows. In the following, we describe the steps involved in our Tabu search method, summarized in Algorithm 2.

Algorithm 2 Sequencing Algorithm

```
1: function SEQUENCER(FlowOrdering(fs))
2:     currentSoln ← GenerateInitialSolution(heuristic)
3:     bestOrder ← currentSoln
4:     while Termination criteria not satisifed do
5:         nhd ← GenerateNeighbourhood(currentSoln)
6:         for each soln in nhd do
7:             computeMkSpn(soln)
8:         SolutionSelection()
9:         currentSoln ← selectedSoln
10:        if selectedSoln better than bestOrder then
11:            bestOrder ← selectedSoln
12:    return bestOrder
```

5.2.1 Initial Solutions

In Tabu search, initial solutions are typically generated using heuristics. Of the many popular heuristics available for NW-JSP, we chose two for generating initial solutions for NW-PSP, viz., the sum of processing times for all the constituent operations of a flow and processing time of the longest operation of a flow. The flows may be ordered using these heuristics in ascending or descending order to create an initial solution. As mentioned in [13], we also used random ordering for obtaining an initial solution. Overall, we execute five runs of the Tabu search algorithm, each starting with a different initial solution—four generated from the two heuristics in ascending and descending order, one based on a random ordering—and choose the best solution from all these runs.

5.2.2 Neighbourhood Generation

After the generation of the initial solution, it is iteratively processed. During each iteration the neighbourhood of the current solution is generated based on its "critical flow". Critical flow of an ordering is that flow which *finishes the last* as per the schedule generated by the time-tabling algorithm, thus, responsible for the current flowspan. In presence of several critical flows, one of them is randomly selected. The principle behind such a neighbourhood function is to turn critical flows of the current solution into non-critical flows, and thus, result in flowspan reduction.

We mainly use two unary operators—*Insertion* and *Swapping*—that operate on a flow for neighbourhood generation of the current solution. The *Insertion* operator removes the critical flow from the ordering and inserts it before the operand flow. While the *Swapping* operator swaps the critical flow with the operand flow. The neighbourhood of the current solution is the set of all possible orderings obtained by execution of the Insertion and Swapping operators on all flows preceding the critical flow in the current ordering. Thus, the neighbourhood contains, at the most, $2(n-1)$ possible solutions.

5.2.3 Solution Selection

After the neighbourhood generation, we select a new solution for the next iteration. To this end, all orderings from the neighbourhood are evaluated using the time-tabling algorithm to determine their flowspan. The ordering with the

lowest possible flowspan in the neighbourhood which does not violate the tabu list, or satisfies the aspiration criterion is selected for subsequent iterations. The tabu list contains a list of flows that were identified as critical flows in previous x iterations. Solutions in the neighbourhood whose critical flow(s) lies in the tabu list are rejected even if they have a low makespan, unless they satisfy the aspiration criterion, i.e., we ignore the tabu list and accept an ordering, if the flowspan of the solution is lower than that of the best ordering encountered so far.

We terminate our execution runs when previous y iterations of the algorithm do not yield an improvement on the best flowspan encountered till the moment. The performance of our algorithm can be tuned with different values of x and y.

5.3 Schedule compression

This procedure of schedule compression is specifically aimed to reduce the number of gate opening events in the TSN schedules. In the NW-PSP, this maps to ensuring that machines (involved with transmission) work in fewer but longer sprints instead of several shorter bursts, i.e., with a gate-open event several packets of scheduled traffic are transmitted. Our idea of schedule compression is based on the principle that start times for certain operations may be delayed such that they end just before the succeeding operations begin on the corresponding machines. In terms of the schedules, it means that the transmission of a scheduled packet may be delayed to a time such that it is finished just before the next scheduled packet for transmission on the same port is available. Overall, this relaxes the no-wait constraint but our algorithm ensures that the flowspan from the original schedule remains unaffected. We explain our approach with a simplified example on a small topology.

Consider the example topology shown in Fig. 5. The machines (switch ports and NICs) of interest are marked as $p_1, \ldots, p_8$. Fig. 6 represents schedules with minimal flowspan for three flows ($F_i : A_i \rightarrow B_i$) in the example topology. We focus mainly on machines responsible for transmission. The schedule shows starting times for all the operations of the flows—transmission delays on switch ports and NICs (marked as $p_{i,s}$ where p_i is the machine operating on the flow) and processing delays (represented as hatched boxes) on the corresponding links. The schedule of the machine responsible for transmission on link p_4 is of interest with respect to the compression algorithm. As can be seen in Fig. 7, there are two gate opening events on machine $p_{4,s}$—the first handles flow F_1 while the second handles flow F_2 and F_3. As shown in Fig. 6 and Fig. 7, the schedule can be modified to delay the transmission of packet belonging to flow F_1 such that a single gate open event can service packets from flows F_1, F_2 and F_3, in quick succession.

It must be noted that this method may not always result in a single gate open event per link like in the presented example. Delaying the transmission of a packet of a flow impacts the starting times of the subsequent operations for the flow. It may so happen that they cannot be delayed, for instance due to impact on the schedule flowspan. Further, while delaying transmission of any packet belonging to a flow, it must be ensured that the order in which the switch processes/transmits it must not be altered. For instance, in Fig. 7, the transmission of packet of flow F_1 on link l_4 cannot be delayed such that packet belonging to flow F_2 is transmitted before it. This would violate the First-In First-Out semantics of queues in switches. Given these constraints, compression of schedules is not a trivial problem.

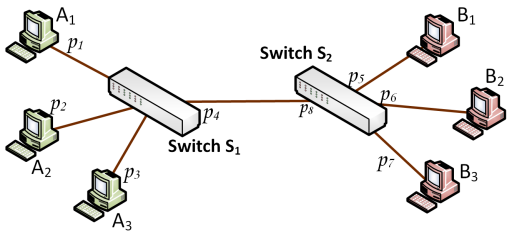


Figure 5: Example topology—6 hosts and 7 links

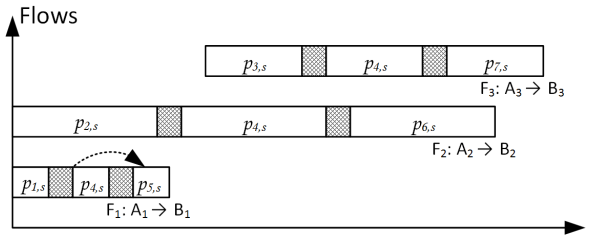


Figure 6: Schedule for 3 flows on example topology. Arrow represents a possibility to compress schedule.

In the following, we present an efficient algorithm for compressing the schedules to reduce the number of gate opening events in the entire schedule. This algorithm can also be used on schedules that are computed using methods other than the presented Tabu search algorithm, for instance the ILP formulations presented in Sec. 4.2. Basically, the scope for schedule compression stems from the existence of "slack" in start times of the operations, as shown in Fig 7. Slack for a given operation is the amount of time by which its start may be delayed on a machine such that it is finished before the next operation is due to start on the machine. For instance, consider a machine that processes operations o_1, o_2, o_3 (belonging to different flows), each needing three time units, between times 1–3, 6–8, 9–11 respectively. Slack for the operations o_1 and o_2 is 2 and 0, respectively, i.e., the start of operation o_1 can be delayed by up to 2 time-units and must start at time $t = 3$ to be finished before o_2 is due to start, while start of o_2 cannot be postponed.

Note that delaying an operation impacts the start times for subsequent operations. Thus, amount of time by which each operation can be delayed clearly depends on the slack of all the subsequent operations of the job. Based on the slack for each operation, the amount of time by which they can be delayed is computable. For a flow with N operations, $\{O_1, O_2, \ldots, O_N\}$, and corresponding slack times

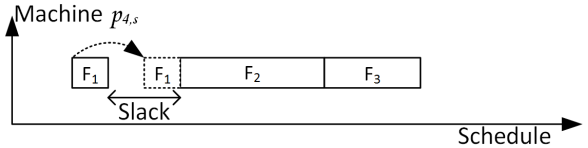


Figure 7: Machine schedule for $p_{4,s}$. Arrow represents a possibility to compress schedule.

$\{S_1, S_2, \ldots S_N\}$, an operation O_i is delayed by time t_i, where $t_i = min(s_i, s_{i+1}, \ldots, s_N)$. Our algorithm, summarized in Algorithm 3, computes slack for all operations of all jobs and then delays their start times correspondingly. It does so iteratively till no further operations can be delayed. The modified schedules have reduced number of gate opening events compared to the original schedule

Algorithm 3 Compression Algorithm

```
1: function COMPRESSOR(Schedule(sched))
2:     compress ← True
3:     while compress do
4:         compress ← False
5:         for each flow do
6:             for each oper do
7:                 computeSlack(sched, flow, oper)
8:         for each flow do
9:             for each oper do
10:                computeDelays(slack)
11:                if delaying possible then
12:                    applyDelays(flow, oper, sched, delay)
13:                    compress ← True
14:    return sched
```

The time required for each iteration of the compression algorithm is directly proportional to the number of jobs (n) and the number constituent operations (N). During each iteration of the algorithm, the start times (after including delays from previous iterations) of at least one operation is finalized. Thus, in the worst case, the algorithm may need up to $N \times n$ iterations. Overall, the worst-case time complexity of our algorithm is $\mathcal{O}(n^2N^2)$. Given that N is bounded, the time complexity reduces to $\mathcal{O}(n^2)$. Thus, our approach provides a solution with polynomial worst-case time complexity for scheduling in time-sensitive networks.

6. EVALUATION

We present the evaluations of our scheduling approach for time-sensitive networks in this section. We evaluated our solutions in various scenarios (randomized topologies with a set of time-sensitive flows randomly generated) to determine the quality of flow schedules they compute (in terms of flowspan), their scalability and the number of gate-opening events they result in. It must be noted that, as described in Section 5, the performance and the worst-case execution times for our approaches do not depend on the type or size of the topologies but rather on the number of flows being scheduled. Nonetheless, we used various models of randomized graphs (Erdős-Rényi (ER) model [9], random regular graphs (RRG) model, and the Barabási-Albert (BA) model [5]) to ascertain this.

6.1 Qualitative Evaluations

In our evaluations, we determine how closely do the schedules computed using the presented approach approximate the optimal schedule in terms of the flowspan. For this we compared the schedules computed using our approach to the ones generated by an ILP solver which solves the ILP formulation of the corresponding problem (cf. Sec. 4.2).

The NW-PSP, an NP-hard problem, however cannot be optimally solved in a reasonable time frame by an ILP solver if the problem instance is large. Our initial attempts to solve small instances with up to 50 flows using CPLEX [1], a state-of-the-art commercial ILP solver from IBM, required over three days to compute optimal solution. However, we observed that the solver quickly generates a feasible solution and improves it subsequently. Further, CPLEX provides primitives to terminate the optimization problem early and obtain the best solution that has been computed till that moment along with the optimality bounds for the solutions, i.e., how does this solution potentially compare with the optimal solution. For our evaluations, we set time-bounds on CPLEX to terminate after a reasonable amount of time and provide the best solution obtained till the moment. We compared this solution with the schedule computed using our approach, in terms of the flowspan.

We executed our evaluations in 30 scenarios on topologies of various sizes (24–100 hosts, 5–20 switches generated using ER, RRG and BA models) with varying number of flows (30–1500). We configured CPLEX with an upper time-bound of 300 min. for solving the ILP formulation for each instance of NW-PSP. Correspondingly, we also computed schedules using our approach for the same instance. For each scenario, we calculated the relative flowspan, i.e, the ratio of flowspan of the schedule calculated by our approach to the flowspan of the schedule computed by the ILP formulations. Fig. 8a shows a cumulative distribution of the computed relative flowspans. Overall, in over 70 % of our scenarios, our approach computes schedules with flowspan equal to or lower than the ones computed by a state-of-the-art ILP solver (restricted to run for 300 min. or less). In about $1/3^{rd}$ of the scenarios, the solutions computed by our approach had a flowspan slightly higher than the ones computed using the ILP. However, in these cases, the difference in the flowspan of these schedules was less than 5 %. Overall, the average relative flowspan for the set of scenarios we evaluated was $\approx$ 97 %. Thus, our approach computes solution which, on an average, have lower flowspan than the ones computed by an ILP solver with a restriction on execution time.

6.2 Scalability Evaluations

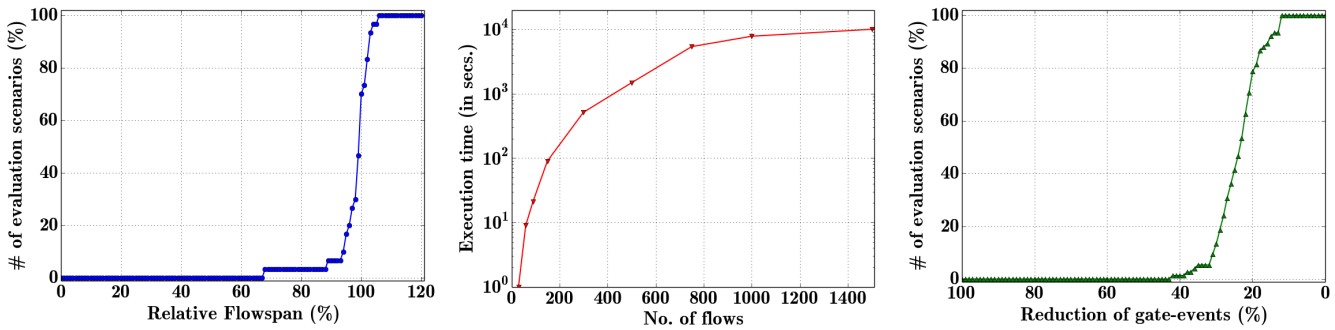
To evaluate the scalability of our approach, we measured the execution times of our algorithm (Tabu Search as well as schedule compression) while computing schedules for varying number (10–1500) of flows. We executed our evaluations on a multi-processor machine (Intel(R) Xeon(R) CPU E3-1245 V2 @ 3.40GHz) with 2×4 cores and 16 GB of memory.

Our evaluations are summarized in Fig. 8b. Up to 50 flows, the scheduling using our approach takes less than 10 secs., while beyond that the execution times increase polynomially, as expected, with the number of flows. Overall, our approach can compute schedules for about 1500 flows in about 3.2 hrs. Moreover, our evaluations suggest that the execution times for our approach are mainly a polynomial function of the number of flows being scheduled. They do not depend on the size of the underlying topology or the model on which the topology is based.

6.3 Impact of Schedule Compression

Compression of flow schedules, as we presented them, is an important aspect of scheduling in time-sensitive networks. To evaluate the effectiveness of our approach, we measured the percentage reduction of gate opening events in the schedules after compression.

For this evaluation, we computed flow schedules in over



(a) Presented approach v/s ILP solution (b) Execution time (c) Impact of schedule compression

Figure 8: Evaluations Results

75 scenarios (varying sizes and models of topologies, varying number of flows etc.). The computed schedules were subjected to our compression algorithm for reducing the number of gate opening events. We not only used the presented Tabu search algorithm but also ILP solvers for computing the initial schedules to show that our schedule compression algorithm is equally effective on schedules computed using different methods. The cumulative distribution function for the reduction in number of gate opening events is shown in Fig. 8c.

Our evaluations show that the schedule compression algorithm, reduces the number of gate-opening events by at least 12 %. In certain scenarios, the achieved reduction goes up to 42 %. Overall, on an average, we observed a reduction of 24 % for gate-opening events in compressed schedules compared to the original schedules.

6.4 Evaluation Summary

In total, our evaluations show:

1. The presented Tabu search algorithm calculates near-optimal solutions for NW-PSP with respect to minimizing the flowspan.

2. The execution times for the presented Tabu search algorithm depends on the number of flows being scheduled. Overall, our approach scales to scheduling over 1500 time-sensitive flows.

3. The presented algorithm for schedule compression results on an average reduction of 24 % for gate-opening events/guard bands.

7. RELATED WORK

The relevance of our work is highlighted by the fact that two major standardization bodies, IEEE and IETF, are working on standardizing enhancements for enabling the IEEE 802.3 networks to handle scheduled traffic. While the IEEE Time-sensitive Networking (TSN) Task Group is targeting enhancements for scheduled traffic in bridges and bridged networks, the Internet Engineering Task Force (IETF) is targeting deterministic data paths with bounds on packet latency, loss, and jitter over Layer 3 routed networks with the IETF DetNets Working Group. Their work has so far resulted in the *IEEE 802.1Qbv* standard which specifies mechanisms for the network elements to implement a given schedule. However, the algorithms for computing schedules using these mechanisms is not in the scope of the standard.

Computing schedules for time-sensitive traffic in real-time networks, like TT-Ethernet, ProfiNET, etc., is reasonably addressed in the literature. In our previous work, we computed transmission schedules by combinedly solving the routing and scheduling problem in a Time-sensitive Software-defined Network [15]. However, we mainly computed coarse-grained schedules for end systems compared to fine-grained link schedules in the current paper. Steiner used Satisfiability Modulo Theories (SMT) to compute a (any) feasible schedule for time-sensitive flows in TT-Ethernet [17]. Extensions of this approach also computes transmission schedules along with task schedules for the tasks executing on the end systems [8][7]. While a feasible solution suffices to deploy a time-sensitive network, in absence of makespan optimization for the schedule, spare bandwidth for best-effort traffic and incremental schedule extensions is affected. Hanzalek et. al. modelled the scheduling problem in ProfiNET as a Resource Constrained Project Scheduling (RCPS) problem to minimize the makespan of the computed schedule [11]. However, this approach is specifically directed towards ProfiNET and does not deal with the problems like guard bands prevalent in time-sensitive networks based on *IEEE 802.1Qbv* standard by means of schedule compression techniques like the one presented in this paper. Further, our Tabu search algorithm is adapted to compute schedules for comparatively larger topologies with an increased number of time-sensitive flows.

We also present related work for accommodating best-effort traffic in real-time networks. In [18], Steiner proposed creation of porous schedules for reserving enough bandwidth of other traffic classes, thus, resulting in networks for supporting systems with mixed criticality. Further, [19] proposes a Tabu search algorithm for adapting schedules of time-triggered traffic such that deadlines for different traffic classes can be satisfied. In contrast, our work focuses only on the schedules for time-triggered traffic, inline with the IEEE standard. We adapt our schedules specifically to reduce number of "gate-open" events for scheduled traffic so that increased capacity is available for other traffic classes by reducing the number of guard bands in the schedules.

8. CONCLUSION

In this paper, we presented the problem of scheduling time-

triggered traffic in time-sensitive networks based on the recently standardized *IEEE 802.1Qbv* standard. We modelled this problem as the No-wait Packet Scheduling Problem (NW-PSP) which can be mapped onto a No-wait Job-shop Scheduling Problem (NW-JSP), a well-known problem from the field of operational research. Our contributions include an efficient meta-heuristic in the form of a Tabu search algorithm to compute schedules by solving the corresponding NW-PSP instance. To further reduce the wastage of bandwidth due to guard bands, we presented a specialised compression algorithm that compresses the schedules to reduce the instances of guard band in schedules. In our future work, we are going to extend our scheduling approach to also account for the transmission period of the time-sensitive flows using the concept of hyper-cycles.

9. REFERENCES

[1] CPLEX Optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

[2] IEEE Approved Draft Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks. pages 1–11, Jan 2015.

[3] IEEE Draft Standard Profile for Use of IEEE 1588 Precision Time Protocol in Power System Applications. pages 1–40, Jan 2016.

[4] IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic. *IEEE Std 802.1Qbv-2015*, pages 1–57, March 2016.

[5] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[6] W. Brinkkötter and P. Brucker. Solving open benchmark instances for the job-shop problem by parallel head-tail adjustments. *Journal of Scheduling*, 4(1):53–64, 2001.

[7] S. S. Craciunas and R. S. Oliver. SMT-based task-and network-level static schedule generation for time-triggered networked systems. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, page 45. ACM, 2014.

[8] S. S. Craciunas and R. S. Oliver. Combined Task- and Network-level Scheduling for Distributed Time-triggered Systems. *Real-Time Syst.*, 52(2):161–200, Mar. 2016.

[9] P. Erdős and A. Rényi. On random graphs I. *Publicationes Mathematicae 6*, pages 290–297, 1959.

[10] F. Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

[11] Z. Hanzalek et al. Profinet IO IRT Message Scheduling With Temporal Constraints. *IEEE Transactions on Industrial Informatics*, 6(3):369–380, Aug 2010.

[12] M. Johas Teener et al. Heterogeneous Networks for Audio and Video: Using IEEE 802.1 Audio Video Bridging. *Proc. of the IEEE*, 101(11):2339–2354, Nov 2013.

[13] R. Macchiaroli et al. Modelling and optimization of industrial manufacturing processes subject to no-wait constraints. *International Journal of Production Research*, 37(11):2585–2607, 1999.

[14] A. Mascis and D. Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.

[15] N. G. Nayak, F. Dürr, and K. Rothermel. Time-sensitive Software-defined Networks for Real-time Applications. Technical Report Computer Science 2016/03, University of Stuttgart, Germany, May 2016.

[16] E. Schemm. SERCOS to link with ethernet for its third generation. *Computing Control Engineering Journal*, 15(2):30–33, April 2004.

[17] W. Steiner. An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In *IEEE 31st Real-Time Systems Symposium (RTSS), 2010*, pages 375–384, Nov 2010.

[18] W. Steiner. Synthesis of Static Communication Schedules for Mixed-Criticality Systems. In *14th IEEE Intl. Symposium on Object/Component/Service Oriented Real-Time Distributed Computing Workshops (ISORCW)*, pages 11–18, March 2011.

[19] D. Tamas-Selicean et al. Synthesis of Communication Schedules for TTEthernet-based Mixed-criticality Systems. In *Proceedings of the 8th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, pages 473–482, New York, NY, USA, 2012. ACM.

[20] E. Tovar and F. Vasques. Real-time fieldbus communications using Profibus networks. *IEEE Transactions on Industrial Electronics*, 46(6):1241–1251, 1999.