Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Algorithms for Calculating Robust Schedules for Time Sensitive Networking (TSN)

Adriaan Nieß

Course of Study:	Informatik
Examiner:	Prof. Dr. Christian Becker
Supervisor:	Dr. Frank Dürr
Commenced:	October 5, 2022
Completed:	April 5, 2023

Abstract

With the 802.1Qbv standard, the time-sensitive networking (TSN) task group has specified a TDMA-based scheduling method based on Ethernet, that allows to ensure real-time guarantees for time-critical data streams. A major challenge here is the computation of the cyclic schedules required for the switch configurations in the network. There exist already a number of different approaches to calculate these schedules. However, most of them focus on supporting a high number of streams and optimize target functions such as the Makespan. This leads to schedules that are particularly fragile against errors in the system model. To close this gap, we have developed a scheduler based on simple temporal networks (STNs) that maximizes the intervals allocated for the transmission of data packets in order to ensure a very high degree of robustness against unforeseen delay or cross-traffic. In a simulation it could be shown that the calculated schedules could still guarantee a loss-free data transmission under compliance with all deadlines if the actual per-hop network delay deviated from the assumed delay by a factor of up to 7. Furthermore, an even larger error did not immediately lead to a breakdown, but to a gradual degradation. This makes the scheduling method presented here as well as the underlying algorithms particularly interesting for applications that come with a high degree of uncertainty in the system model, such as wireless communications or heterogeneous networks.

Kurzfassung

Die time-sensitive Networking (TSN) task group hat mit dem 802.1Qbv Standard einen TDMA basierten Schedulingverfahren auf der Basis von Ethernet spezifziert, mit welchem Echtzeitgarantieren für zeitkritische Datenströme garantiert werden können. Eine große Herausforderung ist hierbei die Berechnung der zyklischen Schedules welche für die Switch Konfigurationen im Netzwerk benötigt werden. Es gibt bereits eine Reihe verschiedener Ansätze um diese Schedules zu berechnen. Die meisten fokusieren sich aber darauf Schedules für eine möglichst hohe Anzahl an Streams zu berechnen und optimieren hierzu Zielfunktionen wie z.B. den Makespan. Das führt dazu, das die Schedules besonders fragil gegenüber Fehler im Systemmodell sind. Um diese Lücke zu schließen haben wir auf Basis von Simple Temporal Networks (STNs) einen Scheduler entwickelt welcher die allokierten Zeitbänder für Datenpakete auf sinnvolle Weise maximiert um eine möglichst hohe Robustheit gegenüber unvorhergesehenem Delay oder Cross-Traffic zu gewährleisten. In einer Simulation konnten gezeigt werden, das die berechneten Schedules auch weiterhin eine verlustfreie Datenübertragung unter Einhaltung aller Deadline gewährleisten konnten wenn der tatsächliche per-Hop Netzwerkdelay um bis zum Faktor 7 vom angenommenen Delay abwich. Darüber hinaus führte ein noch größerer Fehler nicht sofort zu einem Zusammenbruch, sondern einer allmählich kontinuierlichen Verschlechtung. Das macht das hier vorgestellte Schedulingverfahren und die darunterliegenden Algorithmen besonders interessant für Anwendungen in welchen eine hohe Unsicherheit im Systemmodell herscht wie z.B. bei drahtloser Kommunikation oder innerhalb heterogener Netzwerke.

Contents

List of Figures

1 Introduction

The IEEE 802.1Qbv standard describes a time-division multiple access (TDMA) scheme to separate data packets of different traffic classes using a set of transmission gates. By correctly configuring the opening and closing intervals of these transmission gates, upper bounds for delay and jitter can be guaranteed for high prioritized data streams. One of the biggest challenges here is the calculation of schedules for configuring switches. There are already a number of solutions to this problem, for instance the no-wait packet scheduling approach from Dürr[7]. However, this approach assumes constant network delays and no queuing of packets. This is unnecessarily restrictive. In reality, network delays are stochastic. As a result, data packets almost always arrive earlier or later than expected. In addition, it is an unrealistic assumption to begin with to be able to record all system parameters accurately. This becomes particularly apparent in the context of wireless communication. Also in heterogeneous networks with a mixture of different bandwidths, network technologies and hardware, the uncertainty increases. Even though other approaches such as [22] assign flexible intervals to the transmission of data packets, they don't consider that delay can also be underestimated rather than overestimated leading to unintended cascading effects.

In project scheduling, methods to computing flexible schedules based on simple temporal networks (STNs) have existed for quite some time [6]. The goal here is to make the schedules more robust against uncontrollable or unforeseen influences.

In this work, we focused on applying these concepts from operations research to packet scheduling. One of the biggest challenges compared to project scheduling is that schedules must be robust against packets arriving too early. That is to ensure that subsequent frames do not miss their deadline due to queuing delay. Thus we don't intend to queue more than one high-priority frame at a time. With project scheduling, this is not necessary because a subproject that completes earlier than expected, at least in theory, does not affect the progress of other subprojects. In this thesis, we present a novel approach to overcome these limitations by constructing a special quadruple STN from which the constraints as well as the objective functions for a quadratic optimization problem could be derived from. This optimization problem can then be solved to compute a robust schedule. The base algorithm was then combined with further optimizations. For example frame padding was added to ensure that streams are guaranteed to meet their deadline given that the delay assumption are correct. Additionally an IEEE 802.1Qci schedule for ingress policing was computed to ensure that network nodes violating traffic specifications only cause minimal impact. Overall this makes our approach highly robust. We are, to a certain degree, able to compensate for unforseenable delays and interferring traffic. Furthermore by preventing cascading effects due to deadline misses, we can observe a graceful degredation of schedules if the uncertainty in the network exceeds a given threshold.

In the following we will explain the TSN features used to map interval schedules to switch configurations and give an overview on simple temporal networks. We will then explain our robust scheduling approach in more detail. Afterwards, we present our simulation results and finally, we will give an outlook on future developments and possible improvements.

2 Technical Background

This chapter provides an overview of the technologies and methods on which the concepts developed later in this thesis are based upon.

First, different types of network delays are explained. Then, an overview of relevant TSN standards that can be used to map TDMA schedules to Ethernet networks is given. Next, theoretical concepts are explained which we use in our scheduling approach. Namely, linear and quadratic programming, simple temporal networks as well as resource constraints scheduling problems.

2.1 Network Delays

In computer networks, there are generally 4 different types of network delays.

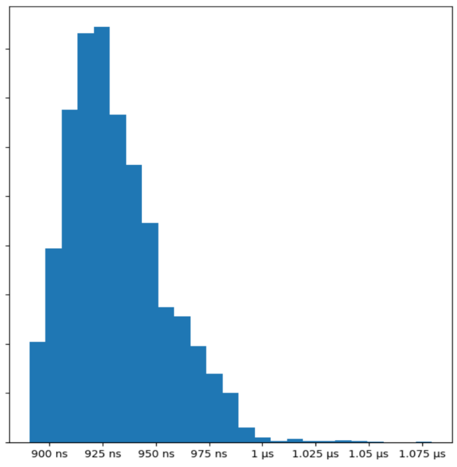
Transmission Delay describes the temporal influence of the bit rate or bandwidth of a medium on the transmission speed of frames. Both the rate at which the voltage levels change and the line code (e.g. Manchester encoding, 4/5 encoding, etc.) have an influence. The higher the bit rate of a channel, the lower the transmission delay.

Propagation Delay specifies how fast a signal can propagate on a physical medium. This depends on the speed of light within the medium and the distance (e.g. length of a cable).

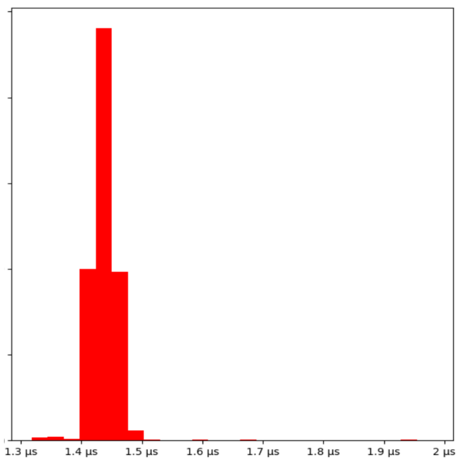
Processing Delay refers to the time needed to process a frame after it has been received completely (store-and-forward) or at least partially (cut-through switching). This time interval includes the calculation of checksums, signing or encrypting frames, inserting timestamps and more. In the remainder of this thesis, we will further subdivide the processing delay to provide a more realistic model.

Queuing Delay can occur whenever the amount of incoming traffic on a bridge exceeds the maximum capacity of an egress port. This can be the case if data streams from several ingress ports are bundled on one egress port or if the data rates of ingress ports exceed those of the egress ports. Otherwise, queuing delays can occur on hosts when applications generate more outbound traffic than can be transmitted or, conversely, the amount of inbound traffic exceeds the rate at which the application can process data. In this paper we focus only on the first type of queuing delay that can occur on switches.

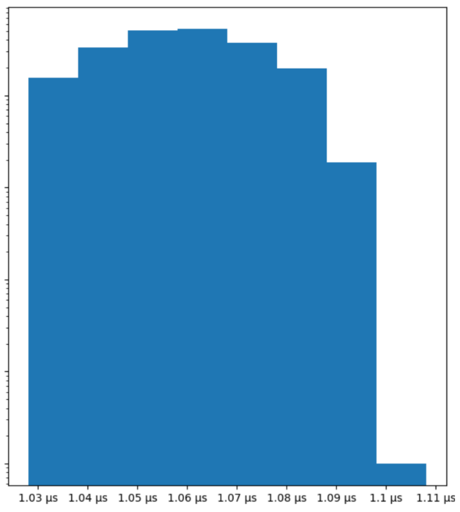
Network delays are unfortunately not very deterministic in reality. Only transmission and propagation delays can be assumed to be static (if we neglect network technologies that dynamically change bandwidth or other exceptions). Processing delays are often subject to greater inaccuracies. An example is shown in figure 2.1 where the processing delays of Ethernet physical interfaces are visualized.



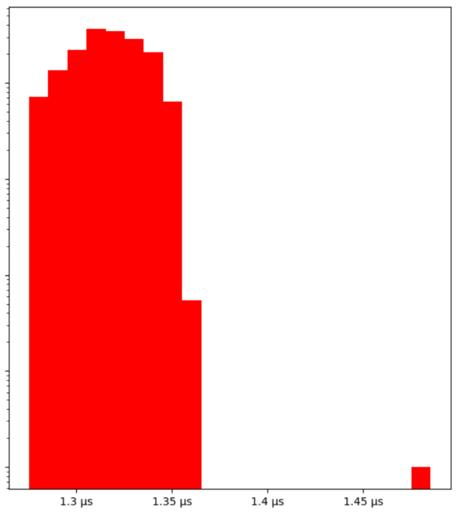
(a) Tx delay, PHY 1



(b) Rx delay, PHY 1



(c) Tx delay, PHY 2



(d) Rx delay, PHY 2

Figure 2.1: Delay distributions of two different sample Ethernet physical interfaces for Tx (SGMII to 100BASE-T1) and Rx (100BASE-T1 to SGMII) directions provided by Keysight[1]. The manufacturers have not been named, but it clearly illustrates the variability between different hardware on the market. Phy 1 comes with a delay range of 890-1079ns on the Tx side and 1318-1981ns on the Rx side. Phy 2 has a delay of 1028-1098ns on Tx and 1275-1477ns on Rx. Very interesting are also the outliers in PHY 2's Rx delay which significantly reduces worst-case latency.

2.2 Time-sensitive Networking

Time-sensitive networking (TSN) is a set of standards being developed by IEEE 802. The TSN task group (TSN TG) which develops theses standards is located next to the task groups for security and maintenance within the IEEE 802.1 working group which focuses primarily on protocol layers above MAC and LLC. Its goal is to provide deterministic connectivity through IEEE 802 Ethernet networks, guaranteed packet transport with bounded latency, low packet delay variation, and low packet loss.

The range of topics on which the TSN TG covers can be seen in figure 2.2. One aspect is high precision clock synchronization through the gPTP protocol specified in the 802.1AS standard[16] which enables synchronization with precision in the range <100ns[9].

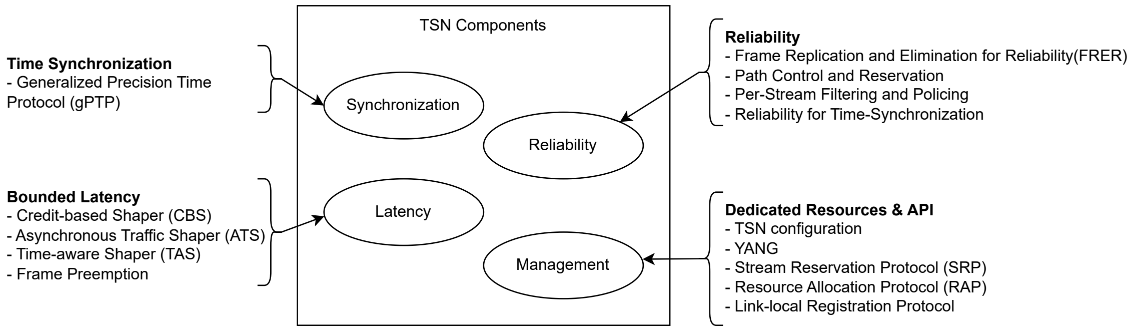

TSN Components
Synchronization
Reliability
Latency
Management
Time Synchronization
- Generalized Precision Time Protocol (gPTP)
Bounded Latency
- Credit-based Shaper (CBS)
- Asynchronous Traffic Shaper (ATS)
- Time-aware Shaper (TAS)
- Frame Preemption
Reliability
- Frame Replication and Elimination for Reliability(FRER)
- Path Control and Reservation
- Per-Stream Filtering and Policing
- Reliability for Time-Synchronization
Dedicated Resources & API
- TSN configuration
- YANG
- Stream Reservation Protocol (SRP)
- Resource Allocation Protocol (RAP)
- Link-local Registration Protocol

Figure 2.2: Overview of Time-sensitive networking

Another component is Reliability. This is addressed by several substandards. Particularly noteworthy are 802.1CB Frame Replication and Elimination for Reliability (FRER), which enables stream identification and the transmission of streams over redundant paths[15], and 802.1Qci Per-Stream Filtering and Policing, which, as the name suggests, enables ingress policing for individual data streams within Ethernet networks[14].

The next component, bounded latency, contains several traffic shapers. The purpose of traffic shapers is to delay frames in order to limit the latency of other frames. Consequently, this means that traffic shaping also increases the average latency. For real-time systems, however, this trade-off is acceptable. Here, the correct functioning is not only determined by the correctness of the output, but the availability of the output at a certain point in time. Traffic shapers make it possible to limit worst-case latencies, which are the critical part for real-time systems, and thus can guarantee correct functioning on the network level. Possible shapers are (among others) the Credit-based Shaper (CBS) specified in 802.1Qav[12], the Asynchronous Traffic Shaper (ATS) from 802.1Qcr[13] and the Time-aware Shaper (TAS) defined in 802.1Qbv[11]. In particular, we will use the TAS to map interval schedules to switches. Therefore, its function will be explained in more detail later.

Last but not least, the TSN TG defines standards for network management, for example to allocate bandwidth for streams along certain paths in the network or to provide uniform YANG data models to enable the configuration of switches via software-defined networking. However, these points are not relevant for this work and we will therefore not discuss them any further.

2.2.1 IEEE 802.1Q Tag

In order to prioritize individual Ethernet frames, it was necessary to extend the Ethernet frame format. The 802.1Q standard therefore introduced the VLAN tag. The tagging mechanism of Ethernet works according to the following principle: If the 2 Byte length field of a frame contains a value greater than 1536 (hexadecimal 0x600), which corresponds to the maximum length of an

Ethernet frame, the field is not interpreted as a length but as a Traffic Protocol Identifier (TPID). The actual frame length must then be read with an offset of the tag size (4 bytes) unless further tags have been inserted.

The VLAN tag is identified by the constant TPID value 0x8100 and contains a 3 bit priority code point field (PCP), a 1 bit drop eligible indicator (DEI) and a 12 bit VLAN ID (VID). The whole structure of a VLAN tagged Ethernet MAC frame is shown in figure 2.3.

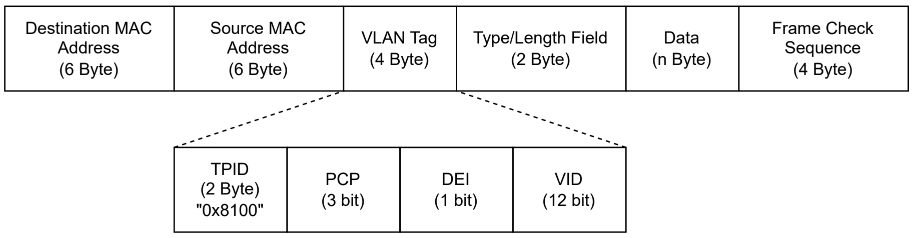

Destination MAC Address (6 Byte)
Source MAC Address (6 Byte)
VLAN Tag (4 Byte)
Type/Length Field (2 Byte)
Data (n Byte)
Frame Check Sequence (4 Byte)
TPID (2 Byte) "0x8100"
PCP (3 bit)
DEI (1 bit)
VID (12 bit)

Figure 2.3: 802.1Q Ethernet frame format

For our scheduling approach it is relevant that with the help of the PCP field a priority value (0-7) can be assigned to each Ethernet frame. In addition, the VID can be used to define multiple logical Ethernet topologies. This allows for example to route frames with the same destination address but different VIDs on different paths.

2.2.2 Stream Identification

Although the 802.1CB standard was published under the title Frame Replication and Elimination for Reliability (FRER), it also specified a very important mechanism for TSN in general, which is not directly reflected in the standard's title - Stream Identification.

The identification of streams, that is the assignment o stream handles to Ethernet frames, is essential for e.g. per stream filtering and policing from 802.1Qci and other TSN mechanisms. Overall, the CB standard specifies two different stream identification functions. An active- and a passive one.

With the passive stream identification function, one or more fields of an Ethernet frame are mapped to a stream handle during ingress (i.e., before the relay). The stream handle is then attached to the frame. TSN components can then read the stream handle. On the egress side, the stream handle is removed again. For the mapping of the stream handle, all fields such as source and destination addresses as well as PCP, DEI and VID fields can be evaluated. In addition, manufacturers could also offer proprietary methods such as deep packet inspection.

The active stream identification function additionally allows to replace fields of the Ethernet header on ingress as well as on egress sides by depending on the stream handle. This functionality is needed for redundant data transmission. Here on ingress side (before the switch relay) a unicast destination address might be replaced by a multicast address to forward a frame on several egress ports. When merging redundant streams the multicast address might then be replaced by a unicast address again. In order to be able to discard duplicated frames in the merge process, the CB standard inserts a special R-tag into frames that contains a sequence number to be able to correlate different frames within a stream. However, the R-tag does not contain the handle itself. This means the stream handle only exists internally within a switch and is not directly visible on the network.

To summarize the stream identification process, we have visualized the data flow once again in figure 2.4.

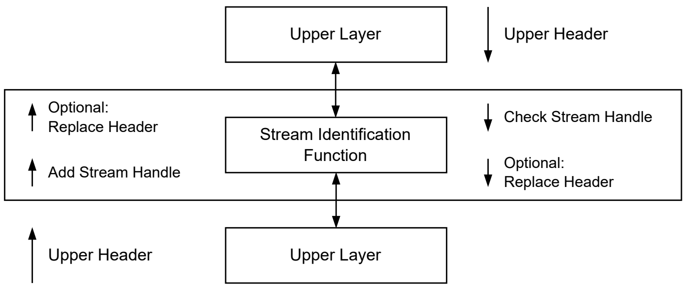


Figure 2.4: Stream identification

2.2.3 TSN Support for Reliability

The TSN standards 802.1CB, 802.1Qci and 802.1AS already offer a number of possibilities for increased reliability of (real-time) data streams within Ethernet networks. See an overview in figure 2.5. With 802.1CB, communication can be protected against link failures, connection failures and to a certain extent device failures through by transmitting frames over redundant paths. 802.1Qci allows protection against blubbling idiots through policing and can eliminate certain error sources that might cause deadline violations. Starting with the 802.1AS-2020 standard, redundant time domains can be used to ensure robustness against crash failures of the network's time source(s).

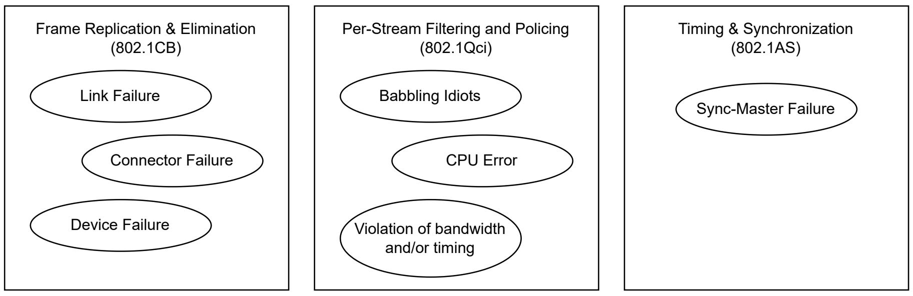

Frame Replication & Elimination (802.1CB)
Link Failure
Connector Failure
Device Failure
Per-Stream Filtering and Policing (802.1Qci)
Babbling Idiots
CPU Error
Violation of bandwidth and/or timing
Timing & Synchronization (802.1AS)
Sync-Master Failure

Figure 2.5: TSN support for reliability

In our scheduling approach, we also employ per stream filtering and policing from the Qci standard. Therefore, we will now explain this in more detail.

2.2.4 Per-stream Filtering and Policing

The IEEE 802.1Qci standard *per-stream filtering and policing* describes how individual streams can be accepted only at specific time intervals using a set of stream gates. If frames arrive outside the accepted times, they are discarded. See figure 2.6. The exact number of stream gates is not specified and may vary depending on the manufacturer. The mapping also allows multiple streams to be assigned to a common stream gate. In our approach, however, we will later use only one stream gate per stream.

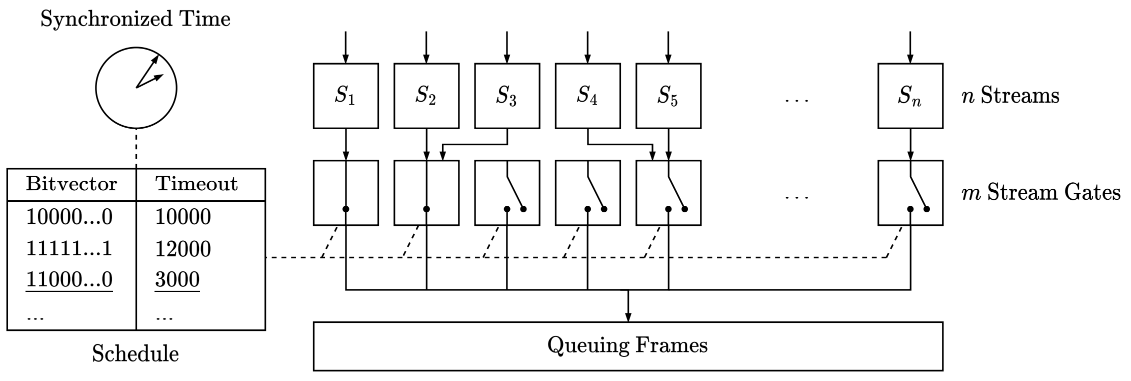


Figure 2.6: 802.1Qci per-stream filtering and policing

In the Qci standard, flow meters are also specified for ingress policing, but we intentionally omitted them in figure 2.6, since they are not used in our scheduling approach.

2.2.5 Time-aware Shaper

After the stream gates there are additional transmission gates which are located on the egress side of a switch (after queuing the frames). These gates are specified in the 802.1Qbv standard and allow traffic shaping and isolation based on a TDMA schedule. The schedule is defined by a set of bitvectors that define when each of the transmission gates is opened or closed as well as a corresponding time interval. To assign the Ethernet frames to queues, the PCP value from the VLAN tag are used. The exact mapping from PCP values to queues can be configured by the network administrator.

Only the queues whose transmission gates are open are used to transmit frames. If no additional shapers have been configured, the frames from the numerically highest, non-empty queue are selected for transmission.

The concept is shown in figure 2.7. For every output port, there can be up to 8 queues. The exact number again depends on the manufacturer. In practice there are at least 2 queues if the time-aware shaper is supported. Otherwise it would not be possible to separate traffic classes.

2.2.6 Generalized Precision Time Protocol

Both per-stream filtering and policing and the time-aware shaper require that all switches in a network are time-synchronized. See figure 2.6 and 2.7. Therefore, the generalized Precision Time Protocol (gPTP) is used to ensure synchronization. This is defined in the 802.1AS standard.

In the network there exists a special network node which serves as a timer for all other nodes. This is also called the grandmaster. The exchange of the sync messages required for the PTP protocol takes place along a distribution tree, also called a sync tree. Both the grandmaster and the sync tree can either be defined statically by the network administrator or determined dynamically using the best master clock algorithm[16].

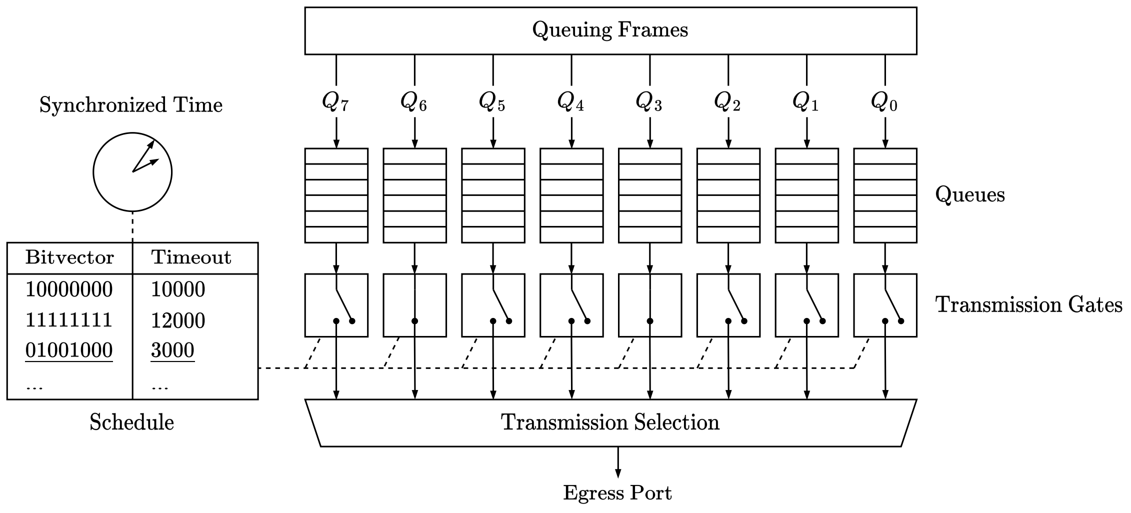


Figure 2.7: 802.1Qbv time-aware shaper

A sync tree is defined by assigning one of the states *master*, slave, *passive* to each port. See figure 2.8 for an example. Sync messages are sent periodically by the Grandmaster. These are then received by gPTP relay nodes via the slave ports and forwarded via the master ports. Between nodes B and C it is necessary to deactivate a link to end up with a valid spanning tree. Therefore the corresponding ports are assigned the passive state.

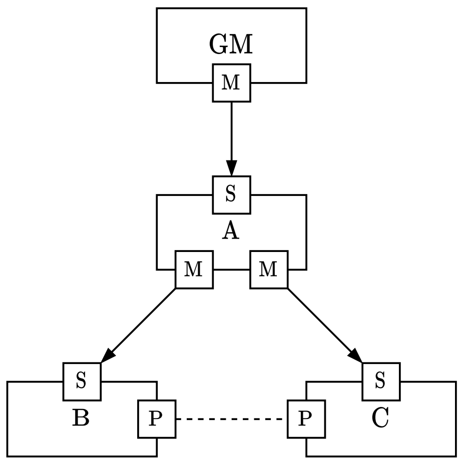


Figure 2.8: Example of a gPTP synchronization tree consisting of 4 nodes with respective port states

The gPTP sync tree is of particular importance because an additional time synchronization error can accumulate between each hop. If you want to determine the maximum time deviation of two hops, it is therefore necessary to consider their distance within the sync tree.

This means that the maximum time error is limited by the network diameter and the time difference between two nodes does not depend on their distance in the network but on their distance in the sync tree. In extreme cases, two directly neighboring nodes can be subject to a relatively high time error if they are far apart in the sync tree. This must of course be taken into account when calculating Qci and Qbv schedules.

2.3 Linear and Quadratic Programming

Linear programming is useful tool for solving scheduling problems. The concept allows to describe linear optimization problems in a canonical form and there are a number of available solvers that can be used to solve them. For example Gurobi, CPLEX or SCIP.

Most often, Integer Linear Programs are used in the context of scheduling problems. However, for the robust scheduling approach presented later, it is sufficient to consider continuous optimization problems which are easier to solve. Therefore, we will only explain the principles of continuous optimization problems here.

Definition 2.3.1 (Linear Program)
An Linear Program (LP) can be described as an objective function to be maximized (or minimized) and a set of linear constraints:

$$\begin{aligned} \textit{maximize} \quad & \mathbf{c}^T \mathbf{x} \\ \textit{subject to} \quad & A\mathbf{x} \le \mathbf{b} \end{aligned}$$

One way to solve linear programs is the simplex algorithm. The algorithm takes advantage of the fact that the feasible region of the LP, if it is not empty, has the form of a polytope. Several constraints (the number is equal to the dimension of the problem) form its faces and individual constraints form the edges connecting the faces. The simplex algorithm tries to find the optimal solution along these edges in the direction of the objective function, similar to the gradient descent method.

The worst-case running time of the simplex algorithm is exponential[17]. E.g. in the well-known Klee-Minty cube example. However, the running time in the average case is polynomial[21], which is why the algorithm performs quite well practice. It's possible to generalize linear programs with quadratic programs with quadratic objective functions (we still assume linear constraints).

Definition 2.3.2 (Quadratic Program)
An quadratic program (QP) can be described as

$$\begin{aligned} \textit{minimize} \quad & \frac{1}{2}\mathbf{x}^T Q\mathbf{x} + \mathbf{c}^T \mathbf{x} \\ \textit{subject to} \quad & A\mathbf{x} \le \mathbf{b} \end{aligned}$$

Well known algorithms such as the simplex algorithm or interior point methods that work for LP can be extended to also work for QP[23][19]. This is because the feasible region is still a polytope. However the optimality conditions have more geometry now and might cause more numerical trouble.

A special case of QP problems are least square problems with $Q = R^T R$ and $c = -R^T \mathbf{d}$ and Q being symmetric positive-definite. One interesting property if Q is positive-semidefinite is that the polytope for the quadratic optimization problem is convex which makes solving these problems easier. In this case it's still possible to employ gradient-descent like algorithms and converge to a global optimum which isn't true for non-convex problems.

Definition 2.3.3 (Least Squares)
Least squares problems can be defined in the form

$$\begin{aligned} \textit{minimize} \quad & \frac{1}{2}||R\mathbf{x} - \mathbf{d}||^2 \\ \textit{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \end{aligned}$$

2.4 Simple Temporal Networks

Simple Temporal Networks were introduced in 1991 by Dechter et al.[6] and have since been used in operations research for efficient computation of flexible schedules. Depending on the literature used, simple temporal networks are also known as simple temporal problems.

Definition 2.4.1 (Simple Temporal Network)
A simple temporal network (STN) consist of a pair $S = (\mathcal{X}, C)$ where $\mathcal{X} = X_1, \ldots, X_n$ is a set of time point variables and C a set of constraints. All of the constraints C_{ij} describe an interval $X_j - X_i \leq c_{ij}$. A solution to a STN is a schedule $\sigma : \mathcal{X} \to \mathrm{R}$ that maps each event $X_i \in \mathcal{X}$ to a non-negative value, such that all constraint in C are satisfied.

STNs are often used to represent task graphs where the variables $\mathcal{X}$ represent possible start times of tasks. Therefore, in the remainder of this thesis, we often refer to the temporal variables of STNs simply as tasks.

2.4.1 Graph Representation

As indicated can STNs be modeled as directed constraint graphs. Here the set of time variables $\mathcal{X}$ form the set of nodes and the constraints $X_j - X_i \leq c_{ij}$ are interpreted as edges $X_i \longrightarrow X_j$ with edge weights c_{ij}. For better readability, the edges $X_i \longrightarrow X_j$ and $X_j \longrightarrow X_i$ are often combined into a single edge which is labeled with an interval which corresponds to the constraint $X_j - X_i \in [-c_{ji}, c_{ij}]$. Unfortunately a disadvantage of the interval representation is that this form is non-canonical, because the direction of interval edges can be chosen arbitrarily. So for implementing algorithms operating on STNs the canonical edge representation is typically chosen while the interval notation is usually easier to read.

Since the representation of lower and upper time bounds can be somewhat confusing at first, we have summarized the different edge representations in table 2.1. Here it is shown how upper- and lower bounds can be defined in a STN and its graph representation. Every STN also comes with a special *temporal reference event*, which start time is typically defined as zero. The start times of all other events are then defined relative to the temporal reference event. In the following we always follow the convention that X_0 is the temporal reference event.

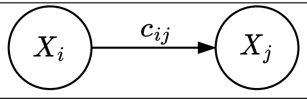
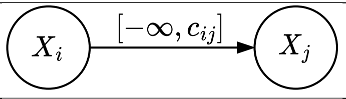
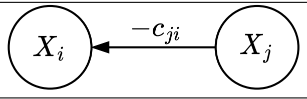
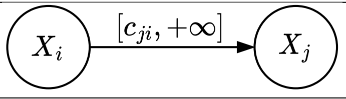
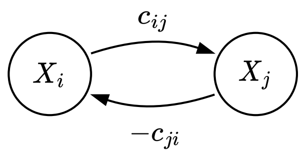
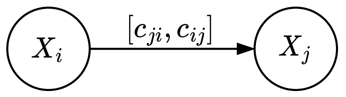
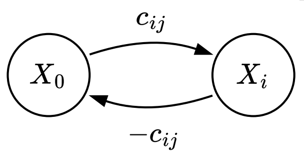
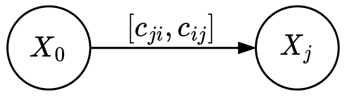
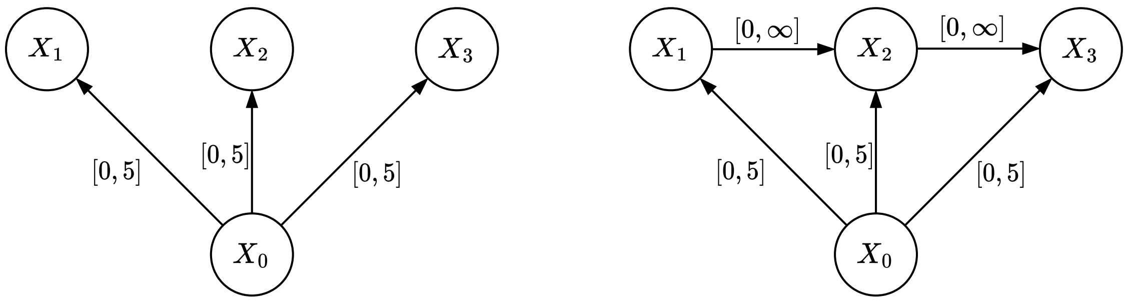
Constraint	Canonical Notation	Interval Notation	Note
$X_j - X_i \leq c_{ij}$			Upper Bound
$X_i - X_j \leq -c_{ij}$			Lower Bound
$c_{ji} \leq X_j - X_i \leq c_{ij}$			Upper and lower bounds
$c_{ji} \leq X_i \leq c_{ij}$			Dep. to temporal reference event

Table 2.1: STN constraints and edge notations

In order to get a better idea of STNs, we have provided two examples which are shown in figure 2.9. In the first example 3 parallel tasks (X_1, X_2 and X_3) are defined with a deadline of 5 time units. In the second example, the 3 tasks are ordered in sequential order. Each task again has a deadline of 5 time units.



(**a**) 3 parallel tasks with deadline of t=5 (**b**) 3 sequential tasks with deadline of t=5

Figure 2.9: Examples of simple temporal networks

2.4.2 Constraint Tightening

The graph representation is very useful to find implicit constraints by interpreting the edge weights c_{ij} on the arcs as length of the path from X_i to X_j. If the constraint set C contains constraints $X_j - X_i \leq c_{ij}$ and $X_k - X_j \leq c_{jk}$ then there exists a path from X_i to X_k via X_j with the combined length $c_{ij} + c_{jk}$. So there exist an implicit constraint $X_k - X_i \leq c_{ij} + c_{jk}$. If implicit constraints are tighter than the already existing constraints, it's possible to replace them with tighter constraints. We call this process constraint tightening.

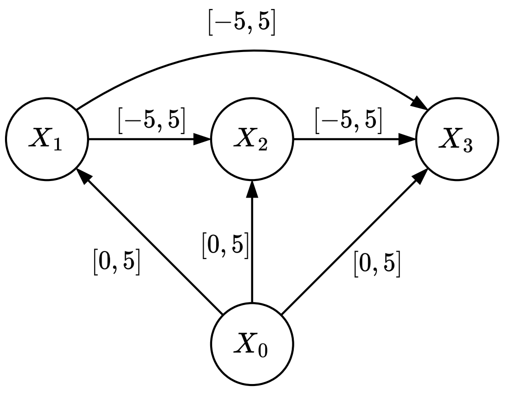
Definition 2.4.2 (Minimal STN)
If all constraints of an STN are tightened as much as possible, the STN is called the minimal STN.

2.4.3 Algorithms

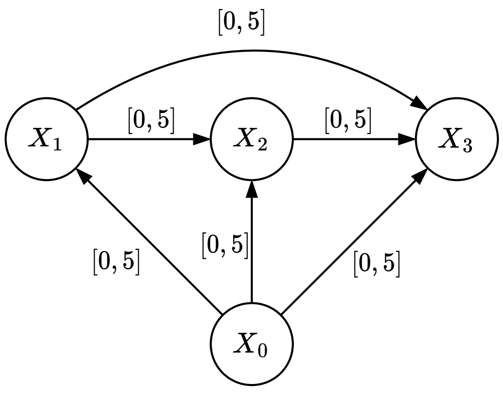
Using the relationship between STN constraints and path lengths, it becomes clear that shortest-path algorithms can be used for constraint tightening. First a distance matrix with all pairs shortest paths must be computed. For this the Floyd-Warshall algorithm[8] could be used which comes with a run-time of $O(n^3)$. Next the minimum STN can then be created from the distance matrix D. For every entry $D[i, j]$ in the distance matrix there exists an edge $X_i \longrightarrow X_j$ in the minimal STN with edge weight $D[i, j]$. Edges with a weight of positive infinity might be omitted.

Since the run-time of all-pairs shortest path algorithms could become a performance bottleneck it makes sense to use more efficient (but more complex) algorithms. For example the Johnson all-pairs-shortest-path algorithm[5] which runs in $O(n^2 \log n + nm)$ using a Fibonacci heap. An algorithm that might perform even better on STNs is Snowball[20] which runs in $O(nmw_d^2)$ with the induced width w_d of a vertex ordering d. If $w_d \in O(\log n)$, Snowball outperforms Johnson's algorithm on general graphs with a run-time of $O(n^2 w_d)$.

Taking the STNs from figure 2.9 as a basis and calculating all shortest paths using one of the algorithms mentioned above, we obtain the minimum STNs shown in figure 2.10.



(a) Minimal STN for parallel tasks



(b) Minimal STN for sequential tasks

Figure 2.10: Minimal STNs derived from examples in figure 2.9

Theorem 1 (Dechter et al.[6])
An STN is consistent if the associated graph does not contain negative cycles.

Proof: Assume an STN contains a negative cycle $C = X_1, \ldots, X_k = X_1$. The sum of inequalities provided by the respective constraints implies $X_1 - X_1 < 0$ which is never true.

When there is no negative cycle, then the shortest path between nodes is well defined. For all node pairs X_i and X_j, shortest paths fulfill the equation

$$d_{0j} \leq d_{0i} + c_{ij} \tag{2.1}$$

This can be rewritten as

$$d_{0j} - d_{0i} \leq c_{ij} \tag{2.2}$$

It becomes apparent that assigning $X_j = d_{0j}$ and $X_i = d_{0i}$ satisfies the constraint $X_j - X_i \leq c_{ij}$. Note that these are the entries $D_S[0,i]$ and $D_S[0,j]$ in the STN's distance matrix. Therefore the tuple

$$X = (d_{01}, \dots, d_{0n}) \tag{2.3}$$

is a solution to the entire STN. Because $X_i - X_0 \leq d_{0j}$ this tuple contains the latest starting times $lst(X_i)$ for all time point variables $X_i \in \mathcal{X}$ assuming that $X_0 = 0$. Analog it can be shown that the tuple

$$X = (-d_{10}, \dots, -d_{n0}) \tag{2.4}$$

contains all earliest start times $est(X_i)$ for the time point variables. □

If only the paths from a single source are of interest or if we are only interesting in checking the consistency of an STN, it is more efficient to use the Bellman-Ford algorithm instead of Johnson or Snowball. This way, the shortest paths can be calculated in $\Theta(nm)$ time. If the Bellman-Ford algorithm is run with the temporal reference event X_0 as the source it will yield a latest start time solution, see equation 2.3, and detect negative cycles in the STN. Note that the negative cycle detection is in fact part of the Bellman-Ford algorithm. This can proof very helpful for heuristics or branch-and-bound methods to abort the recursion at an early stage if an inconsistent STN is detected.

The next theorem turns out to be very useful for incrementally solving STNs. Here one chooses a possible solution for a time variable of the STN which satisfies the constraints. Then one extends this partial solution by successively assigning start times for the other variables until a solution for the complete STN is obtained.

Theorem 2 (Dechter et al.[6])
Assume $S = \{\mathcal{X}, C\}$ is a consistent STN. Every instantiation of a subset $\mathcal{X}_k \subset \mathcal{X}$ of k variables with $1 \leq k \leq n$ that satisfy all the shortest path constraints applicable to $\mathcal{X}_k$ can be extended to all other variables.

Proof: The proof will use an induction on $|\mathcal{X}_k| = k$. For $k = 1$, $\mathcal{X}_1$ consists only of a single variable X_i instantiated by v_i. It must be shown that it's possible to find an assignment $X_j = v_j$ for any other variable X_j so that the shortest path constraint between X_i and X_j is satisfied. For v_j we have to show that the following equation is satisfied

$$-d_{ji} \leq v_j - v_i \leq d_{ij} \tag{2.5}$$

Since consistent STNs do not have negative cycles, we know that $d_{ji} + d_{ij} \geq 0$ and there must exist a value v_j that satisfies equation 2.5.

Next, assuming that the theorem is true for $\mathcal{X}_k$, it must be shown that it also is true for $\mathcal{X}_{k+1}$. Without loss of generality, let $\mathcal{X}_k = \{X_1, \dots, X_k\}$ and $\{X_i = v_i : 1 \leq i \leq k\}$ an assignment that satisfies the shortest path constraints among the variables in $\mathcal{X}_k$. Let $X_{k+1} \notin \mathcal{X}_k$. A value $v_{k+1} = X_{k+1}$ must be found which fulfills all the other shortest path constraints between X_{k+1} and the variables in $\mathcal{X}_k$:

$$-d_{k+1,i} \leq v_{k+1} - v_i \leq d_{i,k+1} \tag{2.6}$$

for $i \in \{1, \ldots, k\}$ or

$$v_{k+1} \le \min\{v_i + d_{i,k+1} : 1 \le i \le k\}, \tag{2.7}$$

$$v_{k+1} \ge \max\{v_i - d_{k+1,i} : 1 \le i \le k\}. \tag{2.8}$$

If we now suppose that the minimum over the set is attained at i_0, the maximum at j_0, we know that v_{k+1} must satisfy

$$v_{j_0} - d_{k+1,i_0} \le v_{k+1} \le v_{i_0} + d_{i_0,k+1}. \tag{2.9}$$

Again, we know that v_{i_0} and v_{j_0} fulfill the constraint between X_{i_0} and X_{j_0}, therefore we have

$$v_{j_0} - v_{i_0} \le d_{i_0,j_0}. \tag{2.10}$$

Because d represents a shortest path constraint, we also have

$$d_{i_0,j_0} \le d_{i_0,k+1} + d_{k+1,j_0}. \tag{2.11}$$

By combining equations 2.10 and 2.11, we end up with

$$v_{j_0} - d_{k+1,i_0} \le v_{i_0} + d_{i_0,k+1}. \tag{2.12}$$

So it can be concluded that there exists at least one value v_{k+1} that satisfies equation 2.9. □

The next corollary is derived from Theorem 2 and describes 2 useful properties that apply to STNs and their schedules.

Corollary 1
Let $S = (\mathcal{X}, C)$ be an STN with a distance matrix D_S. For every $i \in \{1, \ldots, n\}$ let $lst(X_i) = D_S[0, i]$ be the latest starting time of X_i and $est(X_i) = -D_S[i, 0]$ its earliest starting time. For every possible schedule σ for S and every $X_i \in \mathcal{X}$ it holds, that $\sigma(X_i) \in [est(X_i), lst(X_i)]$. Additionally for every $X_i \in \mathcal{X}$ and every $v \in [est(X_i), lst(X_i)]$ it's possible to construct a schedule σ for S such that $\sigma(X_i) = v$.

The corollary shows that STNs can encode a set of schedules. Regardless of which start time is selected for a $v \in [\mathrm{est}(X_i), \mathrm{lst}(X_i)]$, it is always possible to find a schedule in which the associated task is started at the correct time. This makes STNs a more flexible method to describe task execution than classic precedence graphs. Even though STNs only allow the existence of temporal constraints it is possible to transform general resource constraint scheduling problems (RCSP) into simple temporal networks by replacing resource constraints with purely temporal constraints. This has the advantage that schedules for STNs can be calculated more easily and efficiently than for RCSPs and compute flex intervals for every task.

2.5 Resource Constrained Scheduling Problems

A RCSP typically consists of a set of tasks as well as precedence constraint between tasks. This task graph has the form of a directed acyclic graph (DAG). Additionally there is a set of resources. Each of the resources has a finite capacity. Resource constraints describe how much resource instances out of the different resource types are required to complete tasks. We always assume that resources are allocated for the whole operation of a task. It might therefore be useful to split larger tasks into smaller subtasks if resources are only required for a short time span. This means that certain special cases, such as recirculation, cannot be represented with this model. However, it is sufficient for most practical use cases.

In this work we deviate somewhat from the well-known RCSP definitions known from literature. Instead of a DAG for the precedence graph we use a STN. This extends the standard model and allows more flexibility to define temporal dependencies between tasks. Furthermore, this kind of modeling makes it easier to convert RCSP instances into STNs later on.

Definition 2.5.1 (Resource Constraint Scheduling Problem)
A resource constrained scheduling problem (RCSP) consists of a STN $S = (\mathcal{X}, C)$ representing temporal dependencies between tasks. Additional there exists a set of resources $\mathcal{R} = \{R_1, \ldots, R_n\}$, a capacity function cap $: \mathcal{R} \rightarrow \mathbb{N}$ *mapping each resource to its number of instances and a demand function req* $: \mathcal{X} \times \mathcal{R} \rightarrow \mathbb{N}_0$. *A valid schedule σ for a RCSP does fulfill all temporal constraints C from the STN and for each task $X_i \in \mathcal{X}$ and every possible starting time $[est(X_i), lst(X_i)]$, the total resource demand must not be exceed the resource capacity for any resource.*

When defining an RCSP, it must be taken into account that to represent n tasks in the respective STN there must exist n+1 variables due to the additional task/variable for the temporal reference event. Or in other words, there must exists a special start/reference task X_0 which usually has a length of 0 and a fixed starting time of 0.

3 System Model and Problem Statement

In this chapter, the system model for network communication as well as the packet scheduling problem (PSP) to be used as input for robust network scheduling algorithms is defined. Additionally the notion of robustness is introduced which serves as optimization criteria for solving PSP instances.

3.1 Network Model

Existing approaches such as the no-wait packet scheduling problem are not generic enough on the system model to be able to optimize schedules for robustness. Therefore, a more fine-tuned delay model is used here that distinguishes more types of delay and allows for stochastic modeling. Special attention was also paid to take into account non-heterogeneous networks, in which data streams can traverse a combination of various physical media.

3.1.1 Topology

A network is modeled as undirected 3-partite graph $G = (V, E)$. The set of nodes is partitioned into the set of network *nodes* $V_N \subseteq V$, *ports* $V_P \subseteq V$ and physical *media* $V_M \subseteq V$ with $V_N \cap V_P \cap V_M = \emptyset$. There is no explicit differentiation between hosts and bridges. Network nodes and media can only be connected to ports, thus $\forall e \in E : (e \in V_N \times V_P) \vee (e \in V_M \times V_P)$. Using a dedicated node type for physical media, allows the modeling of shared media, as well as redundant links (multigraphs). An example of the mapping from a network topology to a graph is shown in figure 3.1.

The model doesn't explicitly differentiate between hosts and switches, although a host usually comes with only one ingress/egress edge while switches typically have multiple of them.

For time synchronization using gPTP, a spanning tree $G_t = (V, E_t)$ with $E_t \subseteq E$ is used with the grandmaster node $v_{gm} \in V_N$ at the root.

3.1.2 Delay

For the set of ports V_P the processing delays on the RX and TX directions are provided by the mappings

$$\mathrm{Rx} : \; V_P \to \mathcal{N}(\mu, \sigma^2) \tag{3.1}$$

$$\mathrm{Tx} : \; V_P \to \mathcal{N}(\mu, \sigma^2) \tag{3.2}$$

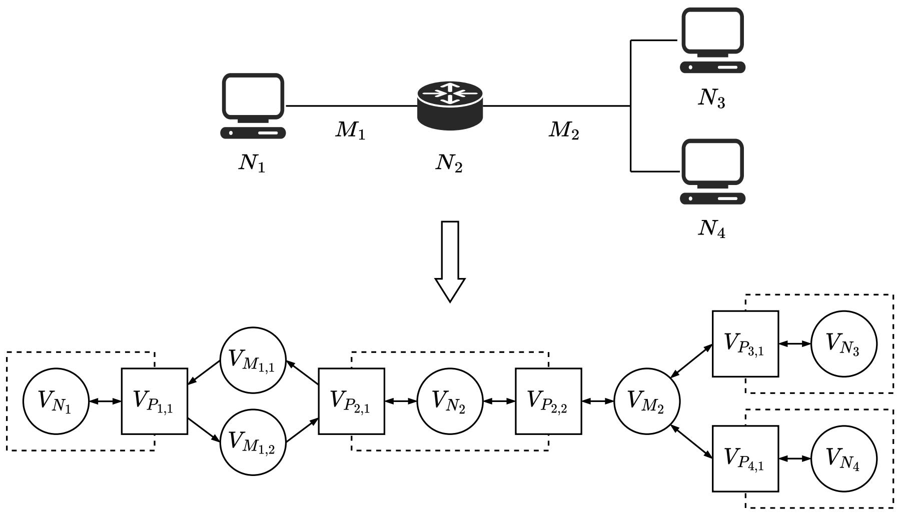


Figure 3.1: Example graph representation of a network. N_1 is connected to the switch N_2 over a full-duplex point-to-point link while N_3 and N_4 are connected via a bus.

These two probability distributions allow to define asynchronous processing delays on a per port basis. This allows modeling the different delays of physical interfaces. Additionally there can be per-node processing delay in the form of bridging delay.

$$\text{Bridging} : V_N \rightarrow \mathcal{N}(\mu, \sigma^2) \tag{3.3}$$

The time synchronization error to the next neighbor is provided by mapping from gPTP slave ports $V_{P_{\text{Slave}}} = \{v_p \mid (v_m, v_p) \in E_t \wedge v_m \in V_M \wedge v_p \in V_P\}$ to the time error between the respective master port (parent node on the gPTP spanning tree). The per-port time error can then later be used to derive a per-node time error.

$$\text{TimeError} : V_{P_{\text{Slave}}} \rightarrow \mathcal{N}(\mu, \sigma^2) \tag{3.4}$$

Certain kind of transport protocols retransmit frames multiple times due to higher bit error rates. To model this behavior an exponential or gamma distributed probability density function *transmissionSuccess* is used.

$$\text{Retransmission} : V_P \rightarrow \Gamma(k, \phi^2) \tag{3.5}$$

For bandwidth and propagation delay of physical media constant mappings are used.

$$\text{bandwidth} : V_M \rightarrow \mathbb{R}^+ \tag{3.6}$$
$$\text{propagation} : V_M \rightarrow \mathbb{R}^+ \tag{3.7}$$

A delay specification D for a topology T is a tuple consisting of the 7 delay functions introduced here.

3.1.3 Streams

High-priority traffic within the network is described as a set of cyclic streams $S = \{s_1, s_2, \dots\}$. We assume every stream to be periodic. Individual streams are described as tuple $s_i = (r_i, d_i, p_i, l_i, E_i)$ with release time r_i, deadline d_i, period p_i, payload length l_i and a distribution tree with $E_i \subseteq E$ that tells us how frames should be routed through the network.

To simplify the model and reduce scheduling complexity we introduce additional constraints:

- Streams are released immediately, thus $\forall s_i \in S : r_i = 0$.
- Deadlines are not greater than their respective periods $d_i \leq p_i$.

3.2 Packet Scheduling Problems

Definition 3.2.1
A packet scheduling problem (PSP) is a 3-tuple consisting of a topology T, a delay specification D, and a set of streams S. Solving a PSP instance means computing sender schedules as well as per-port gate control lists for 802.1Qbv transmission gates to isolate and shape the high priority traffic in such a way that no deadlines are missed.

Although not strictly necessary, it's desirable to also compute a 802.1Qci schedule for every network node to perform per-stream filtering-and-policing on the ingress side. The reason is that traffic shaping is only able to guarantee that deadlines are met if the corresponding data streams comply with their contracts. It therefore makes sense to discard frames if they do not comply with these contracts to prevent them from violating the real-time guarantees of other streams in the short term or even permanently. The last case could occur for example when queue sizes exceed a certain level and cannot fall below that level anymore. That's why for practical applications it always makes sense to combine traffic shaping with ingress policing.

3.2.1 Robustness

The goal of this work is not only to solve PCP instances by computing admissible schedules, but also optimize them for robustness. Overall, we have identified 2 properties that robust schedules shall satisfy.

On one hand schedules should be robust against wrong delay assumptions. In practice it's almost impossible to get all delay assumption exactly right. There could be measurement errors causing wrong estimations, but there could also be more complex error patterns. For example a hardware vendor might change the physical interfaces of an Ethernet device within a production run to compensate for supply shortages which changes the delay properties of the device and invalidate previously calculated schedules. That's why it's desirable to compute schedules with a high likelihood to continue working as expected even if the initial assumptions about the system deviate from the ground truth.

On the other hand there shall also be robustness against interfering traffic. It’s difficult to know all the traffic going through a system in advance. There might be misconfigurations, malfunctions or other reasons for unspecified traffic. Best-effort traffic can be isolated via the 802.1Qbv time-aware shaper, but unexpected high priority frames should also have as little side effects on deadline guarantees as possible.

4 Robust Scheduling Approach

In this chapter, the robust scheduling approach developed in this thesis is explained. The scheduler operates in several steps. At the beginning, a PSP is used as input which is then incrementally transformed into more general problem definitions so that in the end a QP can be set up. Solving the QP will then yield the final schedules which can be deployed switches implementing the IEEE 802.1Q standard.

4.1 Scheduling Pipeline

Figure 4.1 shows the individual scheduler stages. First a PSP is taken as input, for example from an XML file and then converted into a suitable datastructures. Next the PSP is converted to RCSP. This means from now on the scheduler only knows about tasks, resources, temporal dependencies and resource dependencies and does not care about frames, network nodes or ports anymore. In the next step the resource conflicts in the RCSP are resolved using heuristics. After all resource conflicts are resolved there are only temporal dependencies left. This mean from now on the scheduler only knows about an STN. To match the inherent properties of PSPs and to compute an adequate solution it is necessary to translate the STN into a more complex STN which we call Quadruple STN. The reasons and details will be explained later. Afterwards the quadruple STN can be transformed into a QP. This QP can then be solved by a solver (e.g. Gurobi, CPLEX) to retrieve the final schedules.

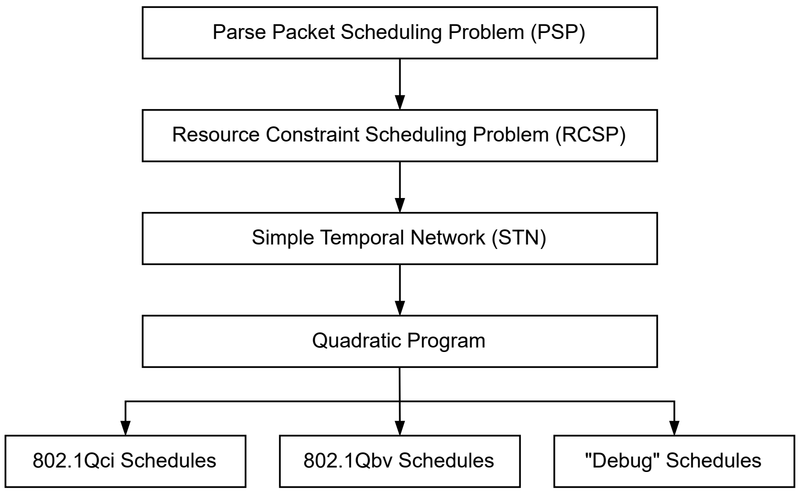

Parse Packet Scheduling Problem (PSP)
Resource Constraint Scheduling Problem (RCSP)
Simple Temporal Network (STN)
Quadratic Program
802.1Qci Schedules
802.1Qbv Schedules
"Debug" Schedules

Figure 4.1: Scheduling pipeline

In the following chapters, the individual scheduler stages are explained in more detail. Before that, however, we will briefly discuss how the scheduler deals with stochastic delays by computing padded task lengths.

4.2 Computation of Task Lengths

Based on the PSP definition it's necessary to find a way to assign time intervals to the sending, forwarding and reception of frames. To generalize these three event types, we simply refer to all of them as frame events. These events also correspond to the tasks when modelling a PSP as RCSP later on. We therefore use the terms frame-event and task interchangeably. The overall goal of the scheduler is to assign valid start intervals to frame events.

Definition 4.2.1 (Frame Event)
A frame event is a tuple $e = (e_{type}, e_{node}, e_{port}, e_{medium}, e_{length})$ with

- *Event type $e_{type} \in \{send, forward, receive\}$*
- *Network node $e_{node} \in V_N$*
- *Ingress port $e_{ingress} \in V_P \cup \{none\}$*
- *Egress port $e_{egress} \in V_P \cup \{none\}$*
- *Physical medium of the egress port $e_{medium} \in V_M \cup \{none\}$*
- *Frame length $e_{length} \in \mathbb{N}$*

The stream in a PSP can be trivially converted to a set of frame events. The different delays that must be taken into account when sending, forwarding or receiving frames, are shown in the exemplary scenario in figure 4.2. Node N_1 and N_3 are hosts, N_2 is a switch. First N_1 sends a frame to N_2. N_2 forwards the frame to N_3.

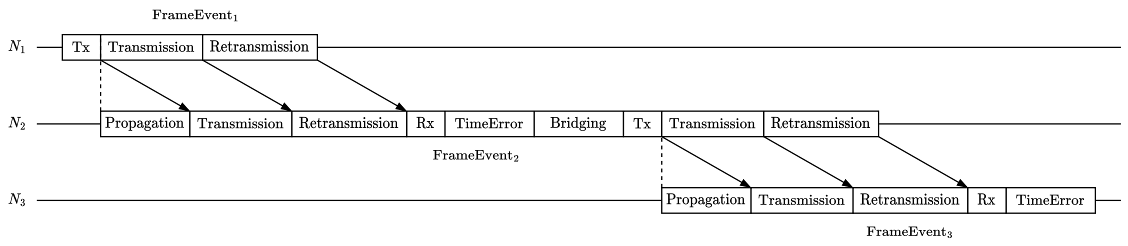


Figure 4.2: Delay scenario

All of the delays shown in figure 4.2 except propagation- and transmission delay are modeled as densities of a random variable. Based on this fact we want to introduce two additional random variables for each frame event e. One random variable *timeBeforeEgress* for the time a frame spends on a node before allocating an egress port. The other random variable *timeOnEgress* is used to describe the time a frame needs to be assigned to an egress port or the physical medium respectively. We define:

$$\text{TimeBeforeEgress}_e = \text{Rx}_{e_\text{ingress}} + \text{TimeError}_{e_\text{node}} + \text{Bridging}_{e_\text{node}} + \text{Tx}_{e_\text{egress}} \tag{4.1}$$

and

$$\text{TimeOnEgress}_e = \text{propagation}_{e_\text{medium}} + \frac{e_\text{length}}{\text{bandwidth}_{e_\text{medium}}} + \text{Retransmission}_{e_\text{egress}} \tag{4.2}$$

4.2.1 Approximating Densities

The random variables *timeBeforeEgress* and *timeOnEgress* can be assumed to be approximately gamma distributed. To compute them we might have to convert some of the normal distributed random variables to gamma distributed ones. This can be achieved by initializing $\Gamma(k,\theta)$ from $\mathcal{N}(\mu,\sigma^2)$ by setting:

$$k = \frac{\mu^2}{\sigma^2}, \quad \theta = \frac{\sigma^2}{\mu} \tag{4.3}$$

From the central limit theorem we know this yields a good approximation if k is big enough. After having converted all random variables to gamma distributed variables, we have to compute the density of the sum of these variables. This density can be approximated with the help of the Welch–Satterthwaite equation and interpreting the gamma distributions as scaled chi-squared distributions.

$$k_{\text{sum}} = \frac{(\sum_i \theta_i k_i)^2}{\sum_i \theta_i^2 k_i}, \quad \theta_{\text{sum}} = \frac{\sum_i \theta_i k_i}{k_{\text{sum}}} \tag{4.4}$$

After approximating *timeBeforeEgress* and *timeOnEgress* with a gamma distribution, we have a closed form solution for the probability density function $f(x)$ and the cumulative density function$F(X)$. The inverse cumulative density function $F^{-1}(x)$ also known as the quantile function $Q(p)$ can be computed numerically by solving an initial value problem with Euler- or Runge-Kutta methods. Additionally some programming libraries such as C++'s Boost library already include approximations for the quantile function.

4.2.2 Frame Padding

Because timeBeforeEgress and timeOnEgress are both non-deterministic, we want to allocate extra time for sending and receiving frames. Therefore we add *ForwardPadding* to our task lengths to be able to handle early frames correctly and *BackwardPadding* to compensate for late frames. In figure 4.3 it can be seen how the probability densities for the transmission start and end of a frame could look like.

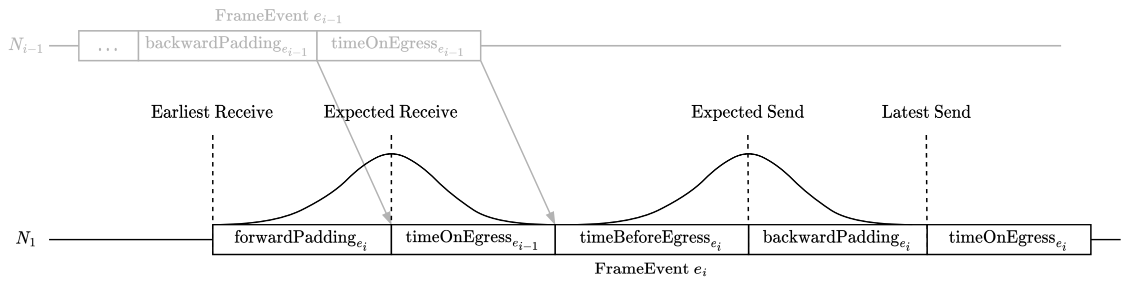


Figure 4.3: Frame padding to compensate for early- or late frames

Due to the linearity of expected values it's easy to compute the expected start- and end times for the transmission of frames. Computing the earliest start and latest end is more complicated. To simplify this computation we assume stochastic independence of delays between hops. For two consecutive frame events $e_i \longrightarrow e_{i+1}$ we define the random variables for the start- and end of frame transmission:

$$\text{Receive}_{e_i} = \text{TimeBeforeEgress}_{e_{i-1}} + \text{propagation}_{e_{i-1}} \tag{4.5}$$

$$\text{Send}_{e_i} = \text{TimeBeforeEgress}_{e_{i-1}} + \text{timeOnEgress}_{e_{i-1}} + \text{TimeBeforeEgress}_{e_i} \tag{4.6}$$

Given the percentils cutoff$^-$ and cutoff$^+$ which might be initialized for example with cutoff$^-$ = 0.001 and cutoff$^+$ = 0.999. With the inverse cumulative distribution functions for *Send* and *Receive* (also known as the quantile function) we define the length of the paddings:

$$\text{forwardPadding}_{e_i} = \mathbb{E}\left[\text{Receive}_{e_i}\right] - F^{-1}_{\text{Receive}_{e_i}}(\text{cutoff}^-) \tag{4.7}$$

$$\text{backwardPadding}_{e_i} = F^{-1}_{\text{Send}_{e_i}}(\text{cutoff}^+) - \mathbb{E}\left[\text{Send}_{e_i}\right] \tag{4.8}$$

Naturally for send- and receive events, the formula has to be slightly adjusted. Senders obviously don't need forwardPadding and receeivers don't need backward padding.

Finally we define the whole task length *taskLength* handling frame events (receiving, processing and transmitting):

$$\begin{aligned}\text{paddedTaskLength}_{e_i} = {} & \text{forwardPadding}_{e_i} + \mathbb{E}\left[\text{TimeOnEgress}_{e_{i-1}}\right] + \mathbb{E}\left[\text{TimeBeforeEgress}_{e_i}\right] \\ & + \text{backwardPadding}e_i + \mathbb{E}\left[\text{timeOnEgress}_{e_i}\right]\end{aligned} \tag{4.9}$$

We conclude that frame padding serves as the first mechanism to make schedules more robust to frames with non-constant delays. However, due to the static nature of the padding, it is only possible to successfully transmit high priority traffic as long as the assumptions about the delay model are correct. That's why we want to additionally assign flexible time intervals later to also protect our traffic against errors in the delay model.

4.3 Resource Constraint Packet Scheduling Problem

After parsing the PSP, the scheduler transforms the PSP into RCSP instance. For this purpose, the frame events of the streams are interpreted as tasks. For each frame event e_i we add a task X_i to the set of tasks $\mathcal{X}$ in the corresponding STN. Additionally, for every port $v_i \in V_P$ we add the resource R_i to the set of resources $\mathcal{R}$ with a capacity $cap(R_i) = 1$. Then for every frame event e_i with $e_{\text{type}} \neq$ receive and every port $v_j \in V_P$ with an edge $(e_{\text{medium}}, v_j) \in E$ we add the resource constraint $req(X_i, v_j) = 1$. That means a send/forward event must allocate every port connected to the medium used to propagate the frame.

If the cycle length of streams do not match it might be necessary to compute a common hyperperiod length first and then add multiple copies of streams that fit into the hyperperiod more than once.

4.3.1 Temporal Dependencies of Consecutive Frames

For consecutive frame events e_{i-1} and its successor e_i belonging to the same stream we must also introduce temporal constraints. We add an edge $X_i \longrightarrow X_{i-1}$ between corresponding tasks to the STN with a weight of $-c_{i,i-1}$ and

$$c_{i,i-1} = \text{paddedTaskLength}_{e_{i-1}} - \mathbb{E}(\text{TimeOnEgress}_{e_{i-1}}) + \text{propagation}_{e_{i-1\text{medium}}} - \text{forwardPadding}_{e_i} \tag{4.10}$$

This edge basically defines a lower bound for the starting time of X_i relative to the starting time of X_{i-1}. We've visualized this in figure 4.4.

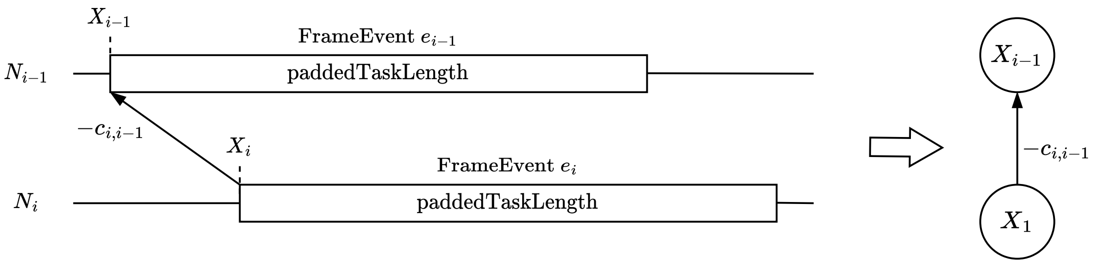


Figure 4.4: Mapping of frame events to temporal constraints

4.3.2 Temporal Dependencies for Start- and End of Stream

In addition to the temporal dependencies of consecutive frames, we must also define a lower bound of 0 for every stream's starting time. So for the first frame event $e_{i,1}$ of every stream s_i and the corresponding STN task $X_{i,1}$ we add a dependency to the temporal reference event by adding the edge $X_{i,1} \rightarrow X_0$ with a weight of 0.

To model the stream deadline of a stream s_i, it's necessary to add a dependency between its last frame event X_{i,m_i} and the temporal reference event. Therefore we add the edge $X_0 \rightarrow X_{i,m_i}$ with a weight of

$$d_i' = d_i - \text{paddedTaskLength}_{e_{m_i}}. \tag{4.11}$$

At this point, an STN with n streams should look as shown in figure 4.5.

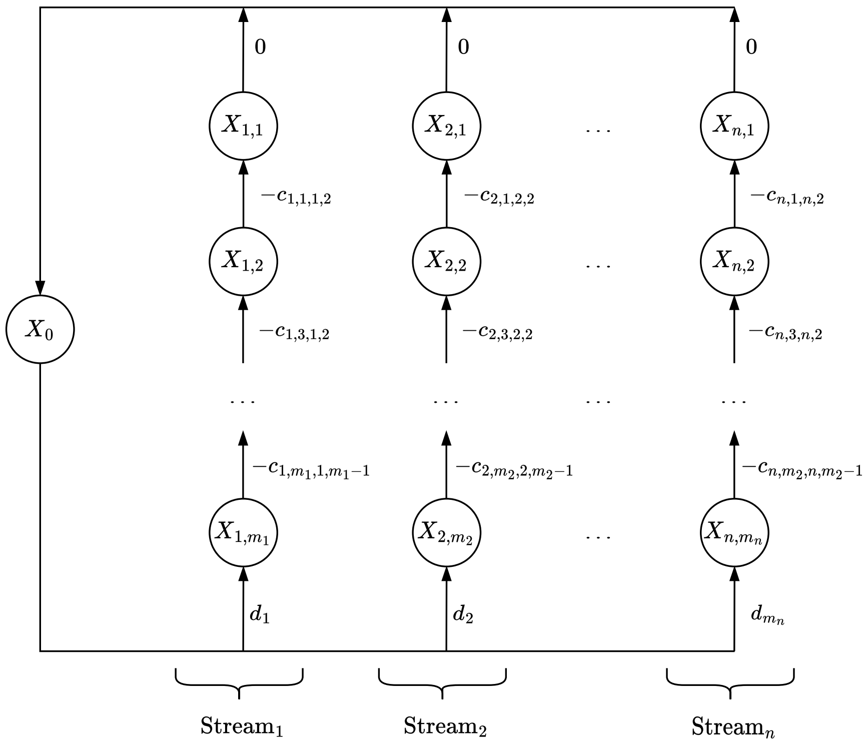


Figure 4.5: Initial STN

4.3.3 Quadruple STN

At the moment, we are only modeling the start times of the tasks using the STN. However, we are also interested in adding flexible intervals as an extension to the static forward and backward padding. We call these intervals *forward-* and *backward flex*. The goal of the scheduler is to distribute these flex intervals over the individual frame events in order to maximize the robustness of the schedules. Since the forward flex grows in negative direction and the backward flex grows in positive direction, we add another time interval to delay frames by buffering them. Intuitively we do not want to delay a frame, because this reduces the frames' flex intervals. In certain situations, however, it can be useful to delay frames if it allows to increase the flex intervals of other frames.

So we split each task in the STN into 4 more tasks and construct a *quadruple STN*. Figure 4.6 shows the added flex- and delay intervals as well as the new task variables and their relationship to these intervals.

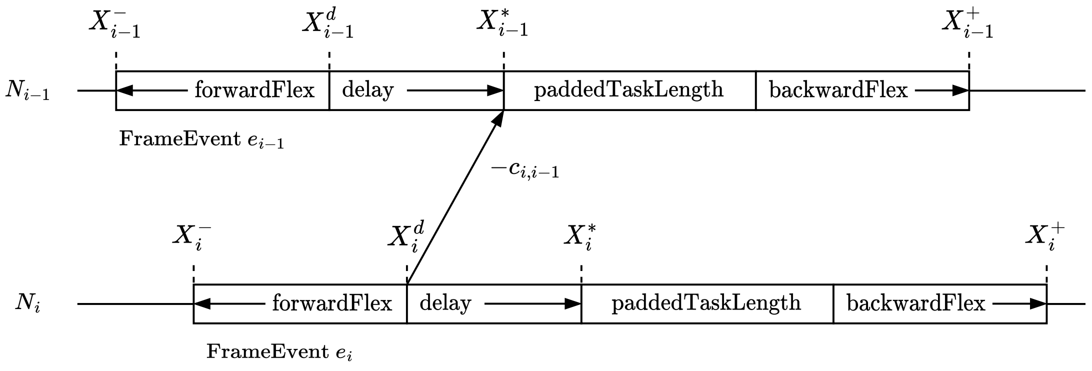


Figure 4.6: Quadruple tasks for consecutive frame events belonging to the same stream.

The quadruple STN is constructed by replacing every task X_i with four tasks X_i^-, X_i^d, X_i^* and X_i^+. Additionally for each task X_i we add the edges $X_i^+ \longrightarrow X_i^*$, $X_i^* \longrightarrow X_i^d$ and $X_i^d \longrightarrow X_i^-$, each with an edge weight of zero.

Every edge $X_0 \longrightarrow X_i$ is replaced with $X_0^+ \longrightarrow X_i^+$ and every edge $X_i \longrightarrow X_0$ with $X_i^- \longrightarrow X_0^-$ (same edge weights).

Temporal constraints $X_i \longrightarrow X_j$ with $X_i \neq X_0$ and $X_j \neq X_0$ are replaced with $x_i^d \longrightarrow X_j^*$ (same edge weight). See figure 4.8.

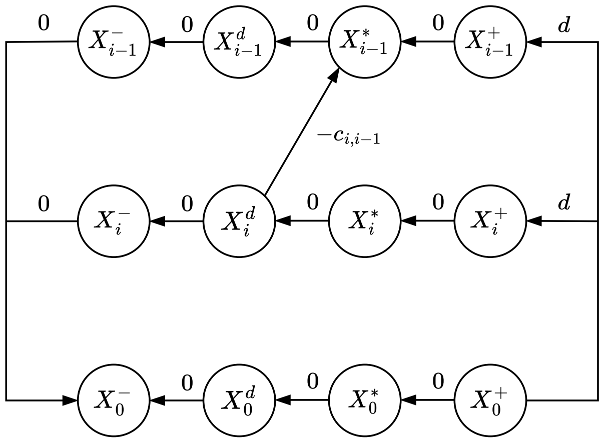


Figure 4.7: Temporal constraint for consecutive frame events in quadruple STN

Using the quadruple STN it's now possible to calculate the earliest and latest starting times for the task variables X_i^-, X_i^d, X_i^*, X_i^+ belonging to frame events e_i. This allows to derive intervals for all these variables. We can use this knowledge to derive heuristics to resolve resource conflicts in a meaningful way and derive an objective function to distribute the flex and delay intervals in the final quadratic program constructed by the scheduler.

4.4 Constraint Posting Heuristics

RCSP instances are NP heavy problems because they generalize the job shop scheduling problem which are proven to be NP-complete [3]. Finding an optimal solution for them is therefore impractical for larger problems. It is therefore necessary to resort to heuristic solutions. One way to solve RCSP instances heuristically is to resolve resource conflicts by adding additional temporal constraints. This method, which is used by our scheduler, is called constraint posting and allows to transform RCSPs into STNs. Finding a fixed time schedule for a pure temporal problem can be achieved in polynomial time[6].

It should be noted that heuristic solutions do not necessarily find a solution if a solution exists. Also finding an optimal solution is not guaranteed. For practical applications, however, approximate solutions that can be found with the help of heuristics are sufficient for the most part.

The heuristics used in this thesis are composed of two parts which are both heuristics themselves. First we use a conflict detection heuristic to find a set of conflicts. In the next step a conflict resolution heuristic selects one of the conflicts. From this conflict two tasks are selected. Next an ordering of these tasks has to be determined and a temporal constraint is inserted into the STN. If this principle doesn't lead to a valid solution, the scheduler either just stops or has to resort to backtracking. In our approach we have not used backtracking so far and only rely on a greedy search strategy. However to be able to pack a lot of streams into a small network, there is no way around backtracking because a greedy search is unlikely to yield a feasible solution (consistent STN).

4.4.1 Conflict Detection

In the conflict detection heuristic the goal is to determine resource peaks. These are basically time points at which a certain set of tasks together exceed the number of available resources of a given type. Overall there are two well-known approaches.

The simplest approach is two use a pairwise method like Cesta et al.[4]. This approach is quite fast, but over-estimates the resource consumption and thus adds more constraints than necessary.

Another approach is computing minimal critical sets. A minimum critical set means that if one task is removed, the set is no longer critical and the resource conflict is solved. If no minimum critical set exists, all fixed time schedules that can be generated are guaranteed to be resource-feasible. Unfortunately it's difficult to compute these sets. The runtime is exponential in the size of the problem. However good approximation algorithms exist[18].

At the moment we have only used the pairwise method to detect resource conflicts. Our evaluation showed that the performance bottleneck lies somewhere else.

4.4.2 Conflict Resolution

For the conflict resolution a conflict set has to be selected first. Usually it makes the most sense to select the biggest conflict. Next a pair of tasks from the conflict set needs to be selected as well as its ordering. Different strategies can be used.

- Tasks can be picked at random. This approach is a good baseline for comparisons with other heuristics.
- Tasks can be prioritized by their latest start time. It's possible to derive the latest start times for every task from the STN determine an ordering. The approach from Alvarez-Valdes and Tamarit[2] is already established in project scheduling and should be expandable for our quadruple STNs.
- The number of successors can also be used to impose an ordering. The paper mentioned above also uses this approach[2].

We only run our scheduler with the random strategy so far. However for plain STNs (without the extension to the quadruple STN) we observed that combining the last two heuristics lead to a runtime improvement of about 10%.

After the heuristic has determined two frame events $e_{k,i}$ and $e_{c,j}$ belonging to streams s_k, s_c and an ordering $e_{c,j} \rightarrow e_{k,i}$ of these events we add the temporal constraint $X^-_{k,i} \rightarrow X^+_{c,j}$ with a weight of zero to the corresponding tasks. This is shown in figure 4.8.

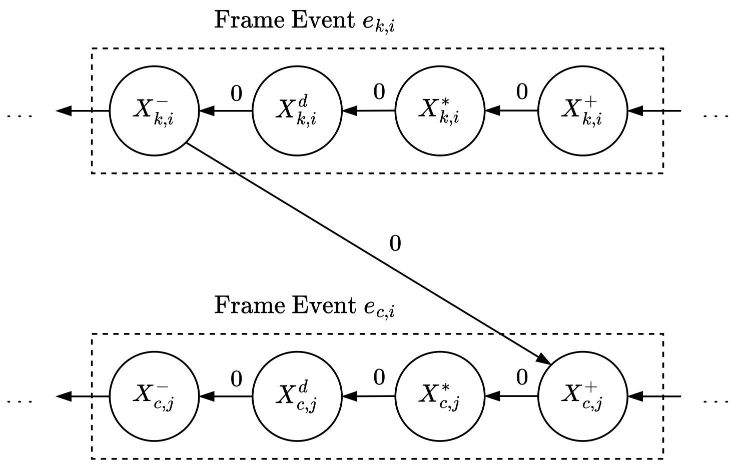


Figure 4.8: Resolution of conflicting frame events

Conflict resolution is applied until the RCSP no longer contains resource conflicts. For this purpose it is necessary to calculate a new distance matrix after each modification of the STN compute new task intervals. Once all conflicts are resolved, it is sufficient to consider only the STN which is a purely temporal problem. Finding a schedule for the final STN is also a feasible schedule for the RCSP.

4.5 Quadratic Program

After the RCSP is converted into a purely temporal problem, the resulting STN can be used to compute a feasible schedule very efficiently. However we are not only interested in a feasible schedule, we also want the schedule to distribute the flex intervals in a meaningful way to frame events. Using a linear objective function and simply maximizing the sum of all flex intervals is not sufficient because one frame could end up with all the flex. That's why we propose two quadratic objective functions. The constraint set X of the final STN can be used directly as the constraint set of the QP.

4.5.1 Distributing Flexibility Equally

The first approach is to distribute flexibility equally by minimizing the quadratic difference of the maximum (forward- and backwards-) flex and the actual flex. Delay is not taken into account in the objective function.

$$\min \sum_i \left(\left(\operatorname{lst}(X_i^d) - \operatorname{est}(X_i^-)\right) - \left(X_i^d - X_i^-\right)\right)^2 + \left(\left(\operatorname{lst}(X_i^+) - \operatorname{est}(X_i^*)\right) - \left(X_i^+ - X_i^*\right)\right)^2 \tag{4.12}$$

Using this strategy yield robust schedules which can be seen in evaluation chapter. Here we used this strategy.

4.5.2 Enhanced Weighting Strategy

The problem with equal weighting is, that delay distributions are usually tail heavy. Therefore we propose an enhanced weighting strategy

$$\min \sum_i w_i^- \left(\left(\operatorname{lst}(X_i^d) - \operatorname{est}(X_i^-)\right) - \left(X_i^d - X_i^-\right)\right)^2 + w_i^+ \left(\left(\operatorname{lst}(X_i^+) - \operatorname{est}(X_i^*)\right) - \left(X_i^+ - X_i^*\right)\right)^2 \tag{4.13}$$

with weights

$$w_i^- = \text{forwardPadding}_i \tag{4.14}$$
$$w_i^+ = \text{backwardPadding}_i. \tag{4.15}$$

This means the assumptions about delay are used for weighting the flex intervals accordingly. If the error deviates only proportionally to the delay assumptions in the system model, this strategy should achieve a very high degree of robustness.

4.6 Mapping Interval Schedules to IEEE Schedules

After the QP has been solved by the corresponding solver, the forward- and backward flex intervals as well as the delay period can be retrieved from the values of the QP variables. These values can then be used to calculate the Qbv and Qci schedules which can be seen in figure 4.9.

The Qbv schedule shown here delays early arriving frames to correct their timing errors by letting them hit a closed transmission gate. This prevents the aggregation of an error when the actual delay is smaller than the assumed delay. Errors caused by late transmissions cannot be corrected unfortunately. The Qci Schedule is chosen to be as large as possible. Only at the end of the interval assigned to the frame the corresponding stream gate must be closed if the remaining time does not allow a successful transmission. This ensures that at any time, no more than one frame is enqueued and at no point in time can enqueued frame delay other high priority frames.

It might be necessary to also add the time needed to transmit one MTU size frame to the closed-period of the stream gate to prevent frames from lower priority queues, that are already in transmission, from interfering. Another option would be adding guard bands on best-effort queues. However if there's enough flex allocated to frame events it also possible to omit this step and save the bandwidth.

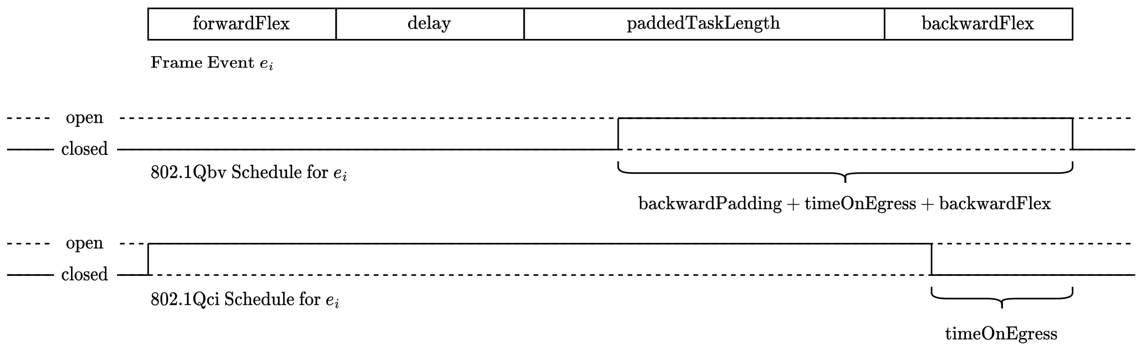

forwardFlex
delay
paddedTaskLength
backwardFlex
Frame Event e_i
open
closed
802.1Qbv Schedule for e_i
backwardPadding + timeOnEgress + backwardFlex
open
closed
802.1Qci Schedule for e_i
timeOnEgress

Figure 4.9: Mapping interval schedules to IEEE 802.1Qbv and 802.1Qci schedules

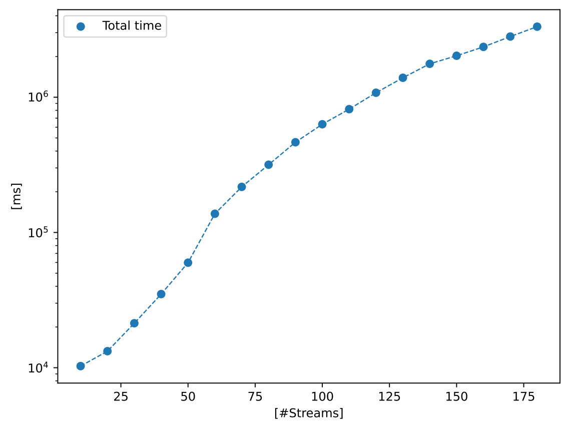
5 Evaluation

The evaluation of the robust scheduler developed in this thesis is split into two parts. In the first part, it is shown how the runtime of the scheduler scales with an increasing number of streams. In the second part, the robustness of generated schedules was investigated. For this purpose, we tested to what extent the adherence to deadlines can be guaranteed when successively increasing errors in the network delays which the schedulers takes as input.

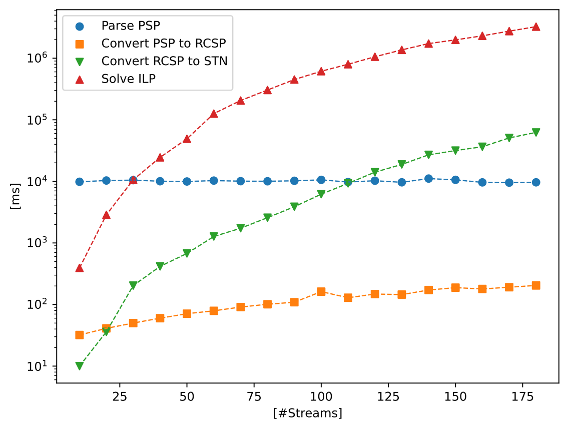
5.1 Performance

For performance evaluation, we generated random networks with 500 nodes and successively added cyclic streams between randomly selected nodes and used a cycle time of 10ms. Our test computer came with a Intel Xeon CPU E3-1231 with 4 cores running at 3.4GHz and 16GB RAM. Additional we used the Ubuntu 22.04.1 operating system. To solve the final quadratic optimization problem we used the Gurobi solver.

In figure 5.1 it can be seen how the scheduler performs on an Erdős–Rényi graph with an average path length of approximately 6 hops. For 150 streams the scheduler took about 33.76 minutes. When looking at the individual scheduling stages it becomes apparent, that solving the quadratic optimization problem takes the most amount of time with a runtime of 33.06 minutes. That is 97% of the total runtime.



(a) Total runtime

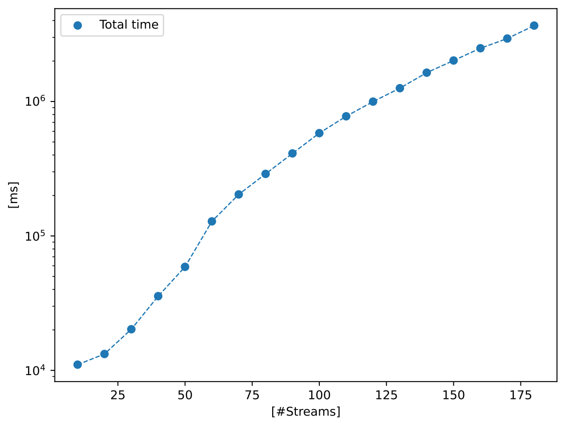


(b) Scheduler stages runtime

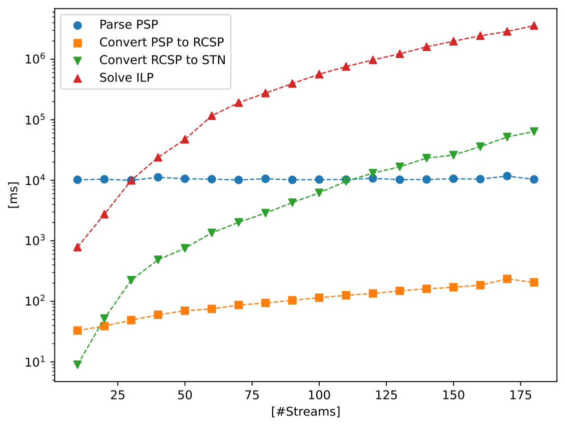
Figure 5.1: Performance evaluation on Erdős–Rényi graph

Running the same experiment on a small-world network (Watts–Strogatz model), also with an average path length of approximately 6 hops, led to similar results. See figure 5.2. For the small-world network we measured a total runtime of 33.64 minutes with 33.02 minutes used for solving the quadratic program.

Overall, we were able to compute schedules with up to 250 streams in an acceptable time (24h) with this approach. However, other approaches are much more capable to compute schedules for a large quantity of streams. The approach of Duerr allows up to 1500 streams[7] and that of Vlk more than 10000 streams[22].



(a) Total runtime



(b) Scheduler stages runtime

Figure 5.2: Performance evaluation on small-world network

We are convinced that there are still a number of ways in which the approach can be optimized further to enable 500-1000 streams. For example, the number of variables in the quadratic optimization problem might be reduced by $\frac{1}{4}$ when exploiting the periodicity of schedules. Instead of calculating forward and backward flex separately, it might be sufficient to assign only a backward flex interval to each frame in the quadratic optimization problem and split this interval later in a postprocessing step. Reducing the number of QP variables might lead to a significant performance boost.

Another optimization opportunity would be the use of better heuristics. This could reduce the number of constraints generated within the QP which would also lead to a reduction in the runtime.

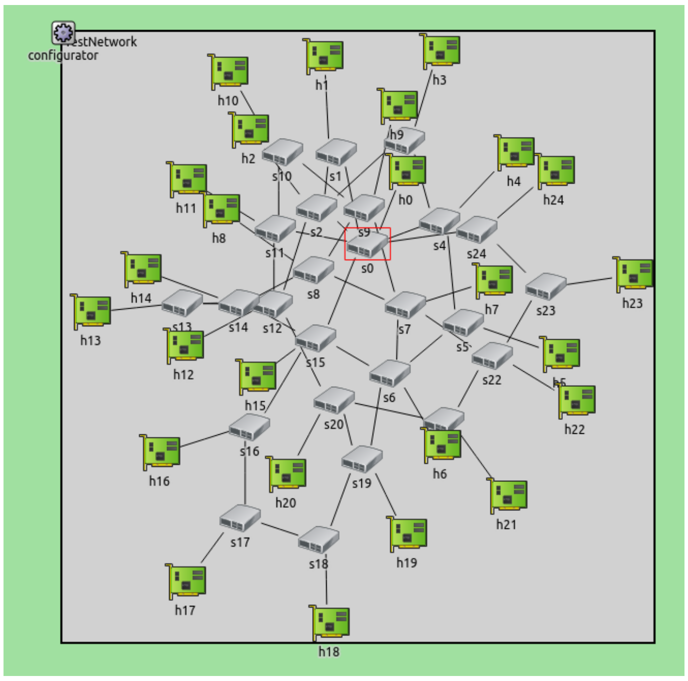
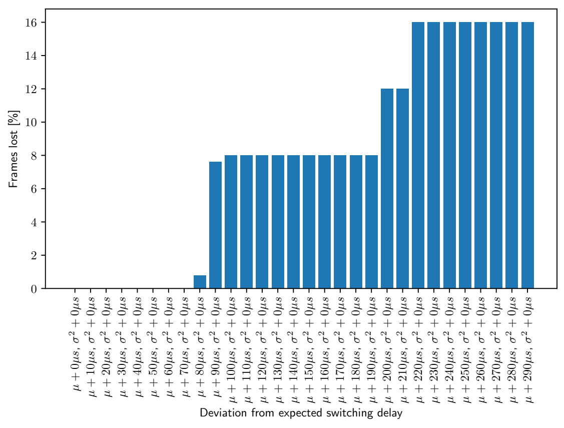


Figure 5.3: OMNeT++ simulation for evaluation of robustness

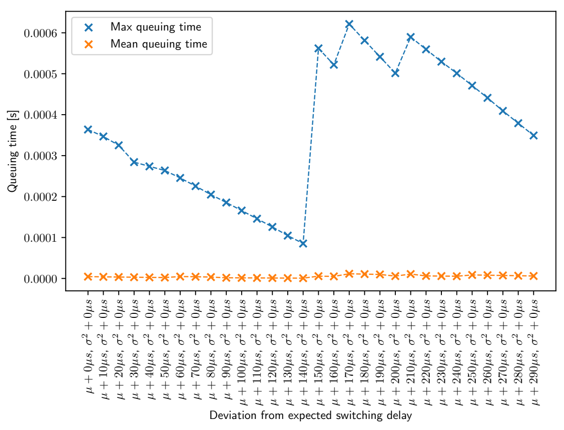
5.2 Robustness

To evaluate the robustness of generated schedules, a random small-world network was generated with 25 host/switch pairs each and 25 streams between random hops. Schedules were then computed assuming a mean switching delay of 10us and a standard deviation of 1us. Based on this assumption we performed a number of simulations using the discrete event simulator OMNeT++. See figure 5.3. We used OMNeT++ version 6.0.1 together with a development branch from the INET framework (commit ID 02eaa3e932) which contains a bugfix in the implementation of the Qci standard.

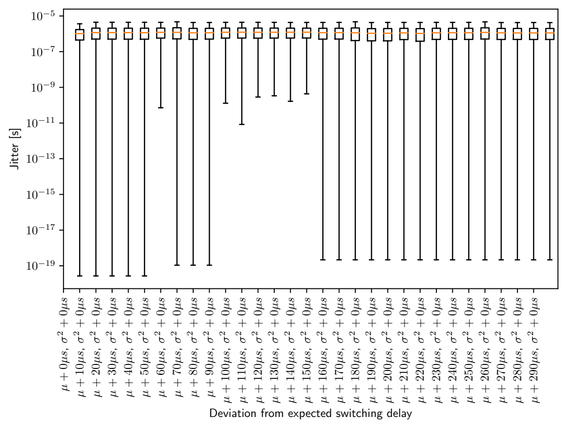
For each simulation, we simulated a time interval of 10s (simulation time). Therefore with a period length of 10ms, each stream was simulated 1000 times. To get an idea of how robust a schedule is, we have now let the real switching delay (in the simulation) deviate step by step from the delay assumed by the scheduler. The result of the simulation can be seen in graph 5.4. It can be seen, that the schedule was able to compensate for switching delay errors up to 70us without lost high-priority frames or deadline misses. Due to Qci schedules used, packet losses of high-priority streams and deadline misses are exactly correlated. The sudden increase in packet loss can be explained by the fact that the stream with the least (backward) flex is affected first. It is therefore particularly interesting for system engineers to look at the bottleneck streams in the generated schedules and, if necessary, to react by adjusting the streams or the topology.



(a) High-priority frames lost (equal to number of deadline misses)



(b) Queuing time of high priority frames

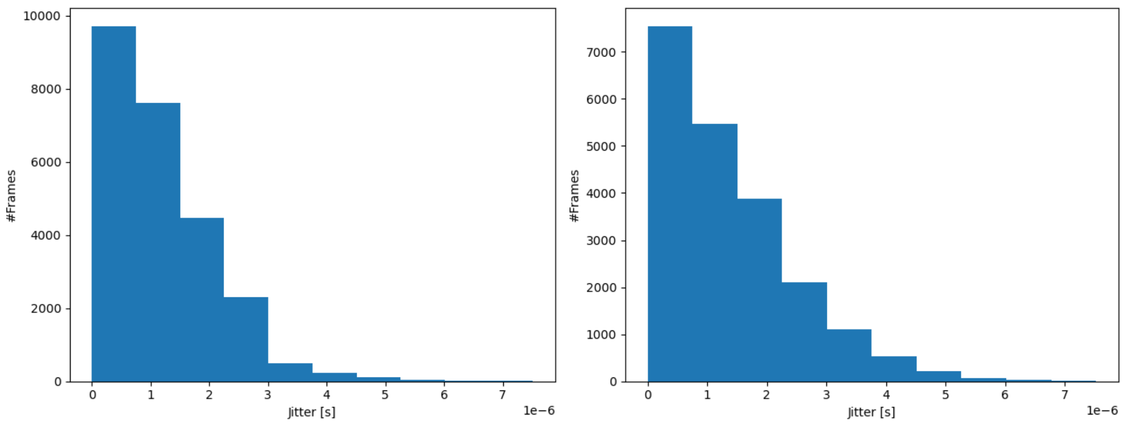


(c) Jitter of high priority frames

Figure 5.4: Variation of mean switch delay

When looking at the queuing delay in figure 5.4b, it becomes apparent that the queuing times of frames decrease before packets are lost. Once packet loss occurs, the maximum queuing times increase rapidly again, since packets that are too late are discarded and thus no longer included in the measurement. The decreasing queuing times themselves can be explained by the fact that the longer the actual delay, the fewer packets arrive within the forward flex or delay intervals.

The same effect as with the queuing delay can be observed with jitter. See figure 5.6c. We have additionally shown in figure 5.5 the histograms of the accumulated jitter over every stream for the first as well as the last simulation run. It can be seen that there is only a slight deterioration, but overall fewer frames are recorded in the last simulation. This is again because frames that miss their deadline are discarded by the Qci schedule and disappear from the histogram.

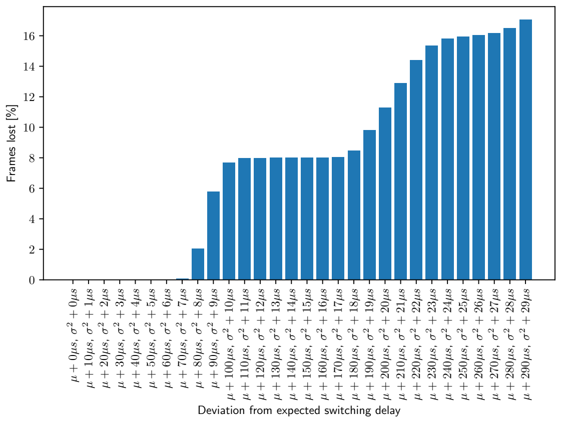


(a) Jitter in the first simulation run ($\mu + 0$, $\sigma^2 + 0$) **(b)** Jitter in the last simulation run ($\mu + 290\mu s$, $\sigma^2 + 0$)

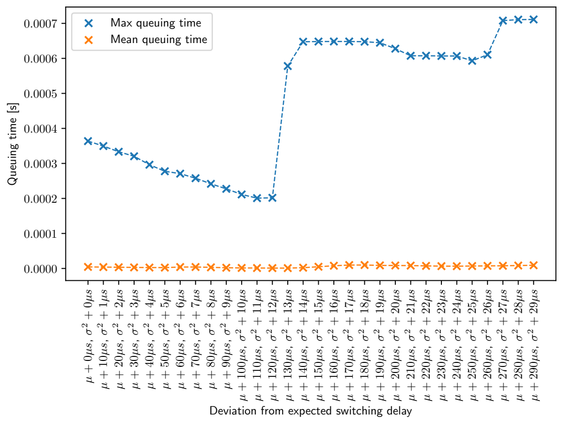
Figure 5.5: Jitter histograms of first and last simulation runs

In a second series of simulations, we did not only adjust the mean switching delay but also its standard deviation. Overall, similar observations were made as before, but the deterioration is less abrupt and more continuous. See figure 5.6.

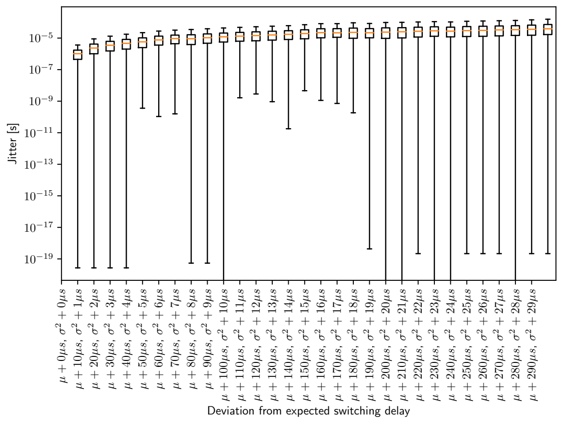
The robustness of the schedules against wrong delay assumptions could thus be proven. Although we have limited ourselves to errors in the switching delay, it can be assumed that the schedules show a similarly robust behavior in the case of deviations in other delays of the system model.



(a) High-priority frames lost (equal to number of deadline misses)



(b) Queuing time of high priority frames



(c) Jitter of high priority frames

Figure 5.6: Variation of mean switch delay and standard deviation

6 Conclusion

It has been shown that with the STN based scheduling approach presented in this thesis schedules can be computed which are robust against erroneous delay assumptions. Additionally schedules are also robust against unforeseen influence of cross-traffic due to per-stream ingress policing. Furthermore, the generated schedules come with a number of other useful properties, such as graceful degredation or the ability to identify bottleneck streams with minimal flex. For practical use, however, it is necessary to further optimize the performance of the scheduler to support a higher number of streams.

6.1 Future Work

- We have measured that the greatest optimization potential of the scheduler lies in simplifying the quadratic optimization problem. We think that, considering the periodicity of the schedules, it could be sufficient to compute only the backward flex using the QP and then split it in a fast postprocessing step. This would reduce the number of QP variables by a quarter.
- At the moment we use ingress policing to isolate high priority streams. That implies that at no point in time more than one frame is enqueued on the highest queue. Intuitively it makes sense to isolate time intervals allocated to different frames, however it might be more efficient to use less restrictive ingress policing to not only allow for temporal flexibility of frame intervals, but also loosening up the frame ordering and allow for *sequential flexibility*. From the perspective of a resource constrained schedulign problem this means that some of the resource conflicts must be resolved dynamically at runtime. The theoretical foundation of this so called *temporal decoupling problem* for STNs was already formulated by Hunsberger[10]. We think that the simplicity of packet scheduling problems as a subset of resource constrained scheduling problems, e.g. resource capacity of 1 for every port, allows solving this problem (at least for full-duplex links) with the use of TSN mechanisms only without the need to deploy software-based controllers on switches.
- The use of branch-and-bound methods seems useful to us, to efficiently compute and examine schedules for densely packed networks. When many streams are placed in a network with few hops, there are more resource conflicts and a greater potential for errors. We are convinced that the use of robust scheduling methods would be particularly interesting here. If only little high-priority traffic is sent through a network anyway, it might not be necessary to resort to such a complex scheduling method in the first place.
- We assumed that the delay between hops is stochastically independent. This is of course a gross simplification. In reality, they are not independent and the delay distributions are more tail-heavy, since only deviations in negative direction can be corrected. Taking this into account the computation of the backward padding could be refined, which would further increase the robustness of the scheduler since the backward padding is also used to weight the backward flex interval in the QP objective function.
- Currently we have approximated delays as a gamma distributions. It could be more efficient to use discretely modeled histograms instead. On one hand this would allow loading measured delay distributions directly into the scheduler instead of having to approximate them by normal- or gamma distributions. On the other hand numerical errors might be reduced. We think that it would be interesting to compare both of these approaches.

Bibliography

[1] Alon Regev and Marty Gubow. *PHY latency and its effects on TSN performance*. 2022. URL: https://standards.ieee.org/wp-content/uploads/2022/12/D1_08_Martin-Gubow-Alon-Regev_PHY-latency-and-its-effects-on-TSN-performance.pptx.pdf (cit. on p. 14).

[2] T. Alvarez-Valdes. "Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis." In: *Advances in project scheduling* (1989) (cit. on p. 38).

[3] J. Blazewicz, J. K. Lenstra, A. H. G. R. Kan. "Scheduling subject to resource constraints: classification and complexity". In: *Discret. Appl. Math.* 5 (1983), pp. 11–24 (cit. on p. 37).

[4] A. Cesta, A. Oddi, V. Marx, S. Smith. "Profile-Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems". In: (Nov. 1999) (cit. on p. 37).

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009. ISBN: 978-0-262-03384-8. URL: http://mitpress.mit.edu/books/introduction-algorithms (cit. on p. 23).

[6] R. Dechter, I. Meiri, J. Pearl. "Temporal constraint networks". In: *Artificial Intelligence* 49.1 (1991), pp. 61–95. ISSN: 0004-3702. DOI: https://doi.org/10.1016/0004-3702(91)90006-6. URL: https://www.sciencedirect.com/science/article/pii/0004370291900066 (cit. on pp. 11, 21, 23, 24, 37).

[7] F. Dürr, N. Nayak. "No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN)". In: (Oct. 2016) (cit. on pp. 11, 41).

[8] R. W. Floyd. "Algorithm 97: Shortest Path". In: *Commun. ACM* 5.6 (June 1962), p. 345. ISSN: 0001-0782. DOI: 10.1145/367766.368168. URL: https://doi.org/10.1145/367766.368168 (cit. on p. 23).

[9] L. Hu, G. Shou, X. Zhang, Y. Liu, Y. Hu. "Multi-domain Time Synchronization Model and Performance Evaluation in TSN". In: *2021 7th International Conference on Computer and Communications (ICCC)*. 2021, pp. 2028–2032. DOI: 10.1109/ICCC54389.2021.9674709 (cit. on p. 15).

[10] L. Hunsberger. "Algorithms for a Temporal Decoupling Problem in Multi-Agent Planning". In: (Oct. 2002) (cit. on p. 47).

[11] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic". In: *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)* (2016), pp. 1–57. DOI: 10.1109/IEEESTD.2016.8613095 (cit. on p. 15).

[12] "IEEE Standard for Local and metropolitan area networks– Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams". In: *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)* (2010), pp. 1–72. DOI: 10.1109/IEEESTD.2010.8684664 (cit. on p. 15).

[13] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks - Amendment 34:Asynchronous Traffic Shaping". In: *IEEE Std 802.1Qcr-2020 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018, IEEE Std 802.1Qcc-2018, IEEE Std 802.1Qcy-2019, and IEEE Std 802.1Qcx-2020)* (2020), pp. 1–151. DOI: 10.1109/IEEESTD.2020.9253013 (cit. on p. 15).

[14] "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing". In: *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)* (2017), pp. 1–65. DOI: 10.1109/IEEESTD.2017.8064221 (cit. on p. 15).

[15] "IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability". In: *IEEE Std 802.1CB-2017* (2017), pp. 1–102. DOI: 10.1109/IEEESTD.2017.8091139 (cit. on p. 15).

[16] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications". In: *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)* (2020), pp. 1–421. DOI: 10.1109/IEEESTD.2020.9121845 (cit. on pp. 15, 18).

[17] V. Klee, G. J. Minty. "How Good is the Simplex Algorithm?" In: 1970 (cit. on p. 20).

[18] M. Lombardi, M. Milano. "A Precedence Constraint Posting Approach for the RCPSP with Time Lags and Variable Durations". In: Sept. 2009, pp. 569–583. ISBN: 978-3-642-04243-0. DOI: 10.1007/978-3-642-04244-7_45 (cit. on p. 37).

[19] K. G. Murty. "A New Practically Efficient Interior Point Method for LP". In: *Algorithmic Operations Research* 1.1 (Jan. 2006). URL: https://journals.lib.unb.ca/index.php/AOR/article/view/93 (cit. on p. 20).

[20] L. Planken, M. M. de Weerdt, R. van der Krogt. "Computing All-Pairs Shortest Paths by Leveraging Low Treewidth". In: *CoRR* abs/1401.4609 (2014). arXiv: 1401.4609. URL: http://arxiv.org/abs/1401.4609 (cit. on p. 23).

[21] D. A. Spielman, S.-H. Teng. "Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time". In: *CoRR* cs.DS/0111050 (2001). URL: https://arxiv.org/abs/cs/0111050 (cit. on p. 20).

[22] M. Vlk, K. Brejchová, Z. Hanzálek, S. Tang. "Large-Scale Periodic Scheduling in Time-Sensitive Networks". In: *Comput. Oper. Res.* 137.C (Jan. 2022). ISSN: 0305-0548. DOI: 10.1016/j.cor.2021.105512. URL: https://doi.org/10.1016/j.cor.2021.105512 (cit. on pp. 11, 41).

[23] P. Wolfe. "The Simplex Method for Quadratic Programming". In: *Econometrica* 27 (1959), p. 170 (cit. on p. 20).

All links were last followed on March 29, 2023.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature