DETERMINISTIC6G

Report on Optimized Deterministic End-to-End Schedules for Dynamic Systems

D3.4

The DETERMINISTIC6G project has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement no 1010965604.

Report on Optimized Deterministic End-to-End Schedules for Dynamic Systems

Grant agreement number: Project title: Project acronym: Project website: Programme:	101096504 Deterministic E2E communication with 6G DETERMINISTIC6G Deterministic6g.eu EU JU SNS Phase 1
Deliverable type: Deliverable reference number: Contributing workpackages: Dissemination level: Due date: Actual submission date:	Public Report D3.4 WP3 PUBLIC 06-30-2024 06-27-2024
Responsible organization: Editor(s): Version number: Status:	USTUTT Frank Dürr, Simon Egger, Lucas Haug v1.0 Final Version
Short abstract:	This report describes the concepts and algorithms for optimizing and dynamically adapting end-to-end schedules with wired and wireless network elements (TSN bridges) to enable deterministic end-to-end guarantees in dynamic environments including mobility, dynamic packet delay, and dynamic stream sets. Different system concepts are proposed to cope with these dynamic effects, such as a YANG data model to describe dynamic stochastic packet delay, different approaches based on the NETCONF protocol to reactively and proactively adapt end-to-end schedules, and a wireless by-pass approach that simplifies end-to-end scheduling and its adaptation. Moreover, several algorithms for calculating and adapting robust end-to-end schedules for scheduled traffic according to IEEE 802.1Qbv (time-aware shaper) are proposed. These algorithms feature the maximization of robustness, fast adaptation through highly optimized algorithms, graceful degradation of the quality of service under increasing packet delay, and schedules optimized for the smooth handover of mobile stations.
Keywords:	time-sensitive networking, TSN, dependable communication, IEEE 802.1Qbv, dynamic wireless systems, 5G, 6G, packet delay, end-to-end scheduling, algorithms, graceful degradation, heuristics, wireless by-pass, YANG, data model, NETCONF, events, adaptation, incremental scheduling, dynamics

Contributor(s):	Joachim Sachs (EDD) Jose Costa Requena (CMC) James Gross (KTH) Gourav Prateek Sharma (KTH) Frank Dürr (USTUTT) Simon Egger (USTUTT) Lucas Haug (USTUTT) János Farkas (ETH) Ferenc Fejes (ETH) Balázs Varga (ETH) Marilet De Andrade Jardim (EAB) Oliver Höftberger (B&R)

Reviewers:	Oliver Höftberger (B&R) Marilet De Andrade Jardim (EAB) Ferenc Fejes (ETH)

Revision History

02/04/2024	Table of contents
04/06/2024	Internal review
20/06/2024	Revision after internal review
21/06/2014	PMT review
27/06/2024	Final version

Disclaimer

Executive summary

Wireless systems, such as 5G or 6G, are dynamic systems by nature. Firstly, packets experience a packet delay while passing through wireless network elements. In contrast to wired networks, this packet delay is stochastic with a relatively large packet delay variation. Moreover, this packet delay is not static but dynamic since it depends on many dynamic factors that influence the wireless transmission such as shadowing by obstacles, scattering and diffraction of the signal, multi-path propagation, fading, etc. On upper layers, these physical effects induce a dynamic bit error rate and frame error rate, which leads to a variable number of retransmissions. Therefore, the probability distribution of the stochastic packet delay is dynamic and might change its mean, variance, etc. Secondly, mobility of end stations intensifies these adverse effects, for instance, due to the requirement to change base stations. Again, this leads to dynamic packet delay distributions. Finally, the set of streams communicated between sender (i.e., talker in terms of Time Sensitive Networking) and receivers (i.e., listeners) is often dynamic. This means, new streams might be added or old streams might be removed at runtime. In particular, new streams to be added dynamically are challenging since this requires the adaptation of the network configuration such as end-to-end schedules to provide end-to-end delay guarantees. This adaptation must be carried out carefully, without violating the guarantees of already admitted streams.

All these dynamic effects pose a big challenge for offering a dependable end-to-end communication service. In this report, we focus on 6G systems implemented as part of a TSN network which consists of wired and wireless bridges. A wireless bridge corresponds hereby to a 6G system (i.e. 6G network and 6G User Equipment (UE), which is in its entirety represented as a virtual TSN bridge, following the TSN integration model specified for 5G [DET23-D31, 3GPP24-23501]. Such virtual (wireless) TSN bridges we refer to as 6GDetCom nodes[1]. We focus on end-to-end scheduling of scheduled traffic according to the IEEE 802.1Qbv standard, which defines a time-driven scheduling mechanism, also known as time-aware shaping. Time-aware shaping requires the calculation of timetables (i.e., schedules) to control the forwarding of packets from egress queues of bridges. Since existing work does often not assume the dynamic effects mentioned above, the problem of calculating robust schedules and adapting schedules to these dynamic effects have not been considered in literature or only with a very limited scope such as dynamic stream sets.

In this report, we present system mechanisms and algorithms for robust and adaptive end-to-end scheduling that can cope with the dynamic effects introduced above. In more detail, our main contributions include:

- A YANG data model to describe dynamic stochastic packet delay. This model provides the essential information about packet delay to the Centralized Network Controller executing the algorithms for calculating and adapting end-to-end schedules.
- Different approaches based on the NETCONF protocol and our YANG data model to trigger dynamic adaption of end-to-end schedules. This includes reactive approaches that “only” react to dynamic changes (break-before-make) and proactive approaches that utilize advanced prediction mechanisms for dynamic packet delay to implement a “make-before-break” approach.

[1] While this report focuses on integrating a future 6G network into a TSN network, the approach is equally applicable to 5G systems.

- We show how to exploit the characteristics of a so-called wireless by-pass that takes advantage of the large reach of wireless links to optimize the network topology.
- Different algorithms to calculate robust and adaptive end-to-end schedules for time-aware shaping. These algorithms enable the maximization of robustness to dynamic packet delay distributions to avoid the costly adaptation of schedules as long as possible, fast adaptation of existing schedules using highly optimized algorithms, and graceful degradation of guarantees provided by schedules under increasing packet delay.

Contents

1 Introduction

This report describes system concepts and algorithms for planning end-to-end packet schedules for dependable end-to-end communication over networks that comprise wired and wireless network elements, including *dynamic* 5G/6G systems. In particular, we focus on dynamic Time-Sensitive Networking (TSN) systems including wireless bridges and wired bridges along the end-to-end communication path between applications (i.e., talkers and listeners), as well as on the adaptation of such systems.

There are different reasons for such a TSN system including wired and wireless bridges to change dynamically and requiring end-to-end adaptation:

- Dynamic stream sets: In many scenarios, it is unrealistic to assume that all streams between talkers and listeners are known a priori, i.e., at system design time, and do not change anymore at runtime. Dynamically adding new streams at runtime requires the adaptation of end-to-end schedules, which is also known as incremental scheduling since streams are added to the schedule incrementally. Such incremental changes must be made carefully in order not to violate the timing guarantees of already admitted streams. Moreover, schedules must be optimized to be extensible in the future.
- Dynamic packet delay characteristics: The packet delay (PD) of wireless bridges is significantly greater than the PD of wired bridges and additionally has greater packet delay variation (PDV). The characteristics of PD can be modelled as non-stationary stochastic processes, i.e., parameters of the stochastic PD distribution such as mean or variance are not constant over time. Or in plain words: The characteristics of non-stationary PD distributions change dynamically over time. Therefore, we also refer to such PD distributions by the term *dynamic PD distributions* in the following to highlight the fact that their parameters are non-constant but dynamic. In order to provide dependable end-to-end communication from the TSN perspective at all times, end-to-end schedules must be adapted to these dynamically changing PD characteristics.
- Mobility: End stations might be mobile. Although mobility is handled within the 5G/6G system and, therefore, transparent (i.e., not “visible”) to the TSN network, mobility might also cause PD distributions to change over time depending on the environment (e.g., obstacles, distance to base station, etc.). As described above for dynamic PD distributions, this requires the adaptation of end-to-end schedules.

It is important to note that end-to-end adaptation of the TSN network is only one possibility of adaptation. Adaptation can also take place within the 5G/6G system (a wireless bridge), e.g., by resource allocation or scheduling of radio resources, i.e., by actively influencing the PD distributions instead of just taking PD distributions as input to the end-to-end scheduling problem in the TSN network. In fact, end-to-end adaptation is a relatively heavy-weight and complex (slow) process, potentially involving many TSN bridges along the end-to-end path and complex computational tasks for calculating end-to-end schedules, whereas the adaptation within the 5G/6G system is affecting only a wireless bridge. In fact, end-to-end adaptation and 5G/6G adaptation complement each other, ideally in a holistic approach. This report is only focused on the end-to-end adaptation of TSN schedules including wireless TSN bridges along the path. Adaptation within the 5G/6G system or a holistic approach integrating end-to-end and 5G/6G adaptation are beyond the scope of this report.

On the one hand, the goal of this report is to present system concepts that enable the adaptation of end-to-end schedules. This includes YANG data models and network control plane interfaces based on the NETCONF protocol for providing dynamic system information – in particular, dynamic packet delay distributions – to a logically centralized network controller performing reactive (when PD distributions have changed) or proactive (based on PD prediction) dynamic re-planning of schedules. Moreover, we describe how to exploit the characteristic properties of wireless communication (in particular its long reach) to implement a so-called wireless by-pass, which reduces the number of end-to-end hops and simplifies the complex calculation of end-to-end schedules. Consequently, the wireless by-pass also simplifies adaption by reducing the time to adapt due to shorter times to calculate new end-to-end schedules.

On the other hand, we present algorithms for (re-)planning end-to-end schedules targeting scheduled traffic according to IEEE 802.1Qbv. Different approaches will be presented for dealing with dynamic stream sets and dynamic packet delay predictions. These algorithms provide different interesting features such as maximizing robustness to dynamic PD distributions, fast rescheduling with dynamic PD distributions and dynamic stream sets, and graceful degradation under degraded channel conditions.

For readers who are not familiar with the DETERMINISTIC6G project, we start with a brief general overview of the project to keep this document self-contained and set the stage for the presentation of the concepts for adaptive end-to-end scheduling. Readers who already know the DETERMINISTIC6G project could skip this sub-section and directly start with Section 0 motivating the need for adaptation to realize dependable end-to-end communication in dynamic systems. Afterwards, we described the relation to other work packages of the project, before giving an overview of the remainder of this document.

1.1 DETERMINISTIC6G Approach

Digital transformation of industries and society is resulting in the emergence of a larger family of time-critical services with needs for high availability and which present unique requirements distinct from traditional Internet applications like video streaming or web browsing. Time-critical services are already known in industrial automation; for example, an industrial control application that might require an end-to-end “over the loop” (i.e., from the sensor to the controller back to the actuator) latency of 2 ms and with a communication service requirement of 99.9999 % [3GPP23-22261]. But with the increasing digitalization similar requirements are appearing in a growing number of new application domains, such as extended reality, autonomous vehicles, and adaptive manufacturing. The general long-term trend of digitalization leads towards a Cyber-Physical Continuum where the monitoring, control and maintenance functionality is moved from physical objects (like a robot, a machine, or a tablet device) to a compute platform at some other location, where a digital representation – or digital twin – of the object is operated. Such Cyber Physical System (CPS) applications need a frequent and consistent information exchange between the digital and physical twins. Several technology developments in the ICT-sector drive this transition. The proliferation of (edge-) cloud compute paradigms provide new cost-efficient and scalable computing capabilities, that are often more efficient to maintain and evolve compared to embedded compute solutions integrated into the physical objects. It also enables the creation of digital twins as a tool for advanced monitoring, prediction and automation of system components and improved coordination of systems of systems. New techniques based on Machine Learning can be applied in application design, that can operate

over large data sets and profit from scalable compute infrastructure. Offloading compute functionality can also reduce spatial footprint, weight, cost, and energy consumption of physical objects, which is in particular important for mobile components, like vehicles, mobile robots, or wearable devices. This approach leads to an increasing need for communication between physical and digital objects, and this communication can span over multiple communication and computational domains. Communication in this cyber-physical world often includes closed-loop control interactions which can have stringent end-to-end KPIs (e.g., minimum, and maximum packet delay) requirements over the entire loop. In addition, many operations may have high criticality, such as business-critical tasks or even safety relevant operations. Therefore, it is required to provide dependable time-critical communication which provides communication service-assurance to achieve the agreed service requirements.

Time-critical communication has in the past been mainly prevalent in industrial automation scenarios with special compute hardware like Programmable Logic Controllers (PLC), and based on proprietary, mutually incompatible wired communication technologies, such as Powerlink and EtherCat, which is limited to local and isolated network domains and which is configured to the specific purpose of the local applications. With the standardization of TSN, and Deterministic Networking (DetNet), similar capabilities are being introduced into the Ethernet and IP networking technologies, which thereby provide a converged multi-service network allowing time critical applications in a managed network infrastructure allowing for consistent performance with zero packet loss and guaranteed low and bounded latency. The underlying principles are that the network elements (i.e. bridges or routers) and the PLCs can provide a consistent and known performance with negligible stochastic variation, which allows to manage the network configuration to the needs of time-critical applications with known traffic characteristics and requirements. Furthermore, using interchangeable TSN hardware components has economic benefits, avoids vendor lock-in and enables third-party support for configuration and troubleshooting.

It turns out that several elements in the digitalization journey introduce characteristics that deviate from the assumptions that are considered as baseline in the planning of deterministic networks. There is often an assumption for compute and communication elements, and also applications, that any stochastic behavior can be minimized such that the time characteristics of the element can be clearly associated with tight minimum/maximum bounds. Cloud computing provides efficient scalable compute, but introduces uncertainty in execution times; wireless communications provides flexibility and simplicity, but with inherently stochastic components that lead to packet delay variations exceeding significantly those found in wired counterparts; and applications embrace novel technologies (e.g. ML-based or machine-vision-based control) where the traffic characteristics deviate from the strictly deterministic behavior of old-school control. In addition, there will be an increase in dynamic behavior where characteristics of applications, and network or compute elements may change over time in contrast to a static behavior that does not change during runtime. It turns out that these deviations of stochastic characteristics make traditional approaches to planning and configuration of end-to-end time-critical communication networks such as TSN or DetNet, fall short in their performance regarding service performance, scalability and efficiency. Instead, a revolutionary approach to the design, planning and operation of time-critical networks is needed that fully embraces the variability but also dynamic changes that come at the side of introducing wireless connectivity, cloud compute and application innovation. DETERMINISTIC6G has as objective to address these

challenges, including the planning of resource allocation for diverse time-critical services end-to-end over multiple domains, providing efficient resource usage and a scalable solution [SPS+23].

DETERMINISTIC6G takes a novel approach towards converged future infrastructures for scalable cyber-physical systems deployment. With respect to networked infrastructures, DETERMINISTIC6G advocates (I) the acceptance and integration of stochastic elements (like wireless links and computational elements) with respect to their stochastic behavior captured through either short-term or longer-term envelopes. Monitoring and prediction of KPIs, for instance latency or reliability, can be leveraged to make individual elements plannable despite a remaining stochastic variance. Nevertheless, system enhancements to mitigate stochastic variances in communication and compute elements are also developed. (II) Next, DETERMINISTIC6G attempts the management of the entire end-to-end interaction loop (e.g. the control loop) with the underlying stochastic characteristics, especially embracing the integration of compute elements. (III) Finally, due to unavoidable stochastic degradations of individual elements, DETERMINISTIC6G advocates allowing for adaptation between applications running over such converged and managed network infrastructures. The idea is to introduce flexibility in the application operation such that its requirements can be adjusted at runtime based on prevailing system conditions. This encompasses a larger set of application requirements that (a) can also accept stochastic end-to-end KPIs, and (b) that possibly can adapt end-to-end KPI requirements at run-time in harmonization with the networked infrastructure. DETERMINISTIC6G builds on a notion of time-awareness, by ensuring accurate and reliable time synchronicity while also ensuring security-by-design for such dependable time-critical communications. Generally, a notion of deterministic communication (where all behavior of network and compute nodes and applications is pre-determined) is extended towards dependable time-critical communication, where the focus is on ensuring that the communication (and compute) characteristics are managed in order to provide the KPIs and reliability levels that are required by the application. DETERMINISTIC6G facilitates architectures and algorithms for scalable and converged future network infrastructures that enable dependable time-critical communication end-to-end, across domains and including 6G.

As mentioned above, the wireless systems that are considered in DETERMINISTIC6G have stochastic and dynamic behavior. To cope with these challenging properties, the system must be able to adapt. In this report, the focus is on the adaptation of end-to-end schedules according to the IEEE 802.1Qbv TSN standard [IEEE15-8021Qbv]. YANG data models are presented to describe and provide dynamic packet delay, which can be integrated into existing TSN standard YANG models. The NETCONF protocol is applied to transfer dynamic delay information from bridges and to trigger the adaptation of end-to-end schedules reactively and proactively using packet delay prediction concepts from the project. We describe how to exploit the properties of wireless communication to implement a so-called wireless by-pass, which by-passes several wired hops, which also simplifies the complex task of end-to-end scheduling. Moreover, optimized algorithms are presented to calculate and adapt end-to-end schedules for dynamic stream sets and for dynamic PD distributions, which induce novel characteristics such as maximization of robustness to dynamic PD distributions, fast re-scheduling with dynamic PD distributions and stream sets, and graceful degradation under degraded channel conditions.

1.2 Background on Adaptation of End-to-End Scheduling

As already mentioned above, there are different reasons for a TSN system consisting of wired and wireless bridges to change dynamically and requiring the adaptation of end-to-end scheduling. One reason is the need to dynamically add or delete streams between talkers and listeners. For instance, the popular "plug & produce" paradigm envisions that network entities (i.e., individual device, machine modules, or whole machines) acting as talkers and listeners, can be added or removed dynamically at runtime to a factory shop floor. These network entities should be integrated automatically into the system when connected ("plugged in") without interrupting the production process, i.e., without first stopping, then reconfiguring, and finally restarting the system. Instead, the system must be reconfigured "on-the-fly" without stopping on-going communication and services. From a TSN perspective aiming for dependable real-time guarantees such as meeting deadlines reliably, this means that schedules at bridges controlling the timely forwarding of real-time traffic need to be adapted and deployed at runtime. So-called incremental scheduling approaches deal with the problem of incrementally admitting new streams and adapting an existing schedule to accommodate new streams (or remove old streams) without compromising the timing guarantees for already admitted streams. The incremental scheduling problem has already received some attention from industry and the research community in the field of TSN (see Section 3.2 for an overview of related work). However, we will revisit this general problem in this report in the light of the specific properties of wireless TSN systems, in particular, probabilistic PDs, requiring *robust* incremental scheduling that can deal with uncertainty.

Another reason requiring adaptation, specifically in TSN networks including wireless bridges, is dynamic PD distributions. In wireless systems, packet delay follows a stochastic PD distribution. Typically, the PD distribution of wireless bridges has a significantly greater packet delay variation (PDV) and is heavy-tailed, i.e., relatively large PD values overshadow other sources of uncertainty in wired systems, where the probability values of the tail of the distribution might decrease exponentially. Additionally, the PD distribution might change dynamically depending on the physical environment causing adverse effects such as shadowing, reflection, diffraction, scattering, and slow fading of the physical signal. The dynamic quality of the physical signal influences properties such as the bit error rate and frame error rate, which in turn affect other mechanisms on higher layers such as the required number of retransmissions to correctly receive a frame eventually. Altogether, this will manifest itself in variable packet delay. Consequently, we cannot assume that the PD distribution is static. This has a great impact onto end-to-end scheduling, which should ensure dependable communication in any case. To deal with dynamic changes in PD distributions, we need several steps:

- First of all, the entity calculating end-to-end schedules must be made aware of the stochastic PDs. According to one common model of network configuration in TSN, we assume that a Centralized Network Controller (CNC) is in charge of calculating end-to-end schedules and configuring all bridges along the path, based on a global view onto the network and streams. To communicate stochastic PD information from bridges to the CNC, extended data models are required beyond what has so far been defined in standards to model static worst-case delay. To this end, we design and present extended YANG data models for describing PD. Moreover, we present how this information can be transmitted over the standard NETCONF protocol from bridges to CNC, either using query mechanisms or event-based communication.

- Secondly, new algorithms for re-planning schedules based on the provided PD distributions are required. This dynamic re-planning of schedules is a big challenge since calculating schedules is, from a mathematical perspective, a very complex task, formally, often an NP-hard problem. Therefore, theoretically well-founded approaches are required to quickly generate new schedules in time. Furthermore, prediction approaches for stochastic PDs are very useful since they can trigger proactive schedule re-planning (make before break) instead of reacting when the old schedule is already compromised (break before make). Moreover, we present a scheduling approach that allows for quickly calculating new schedules based on an old schedule (instead of starting from scratch) and that gracefully degrades the streams' timeliness and reliability guarantees for worsening channel conditions instead of steeply dropping to no guarantees.

Finally, we also consider how to exploit the characteristic properties of wireless communication to implement a so-called wireless by-pass. Due to the long reach of 6G networks, many wired hops can be by-passed. The reduced number of hops simplifies end-to-end scheduling and, in turn, also allows for faster adaptation due to a reduced time to calculate new end-to-end schedules.

1.3 Contributions of the Report

The overarching goal of this report is to present system concepts and algorithms to support dependable end-to-end communication in mixed wired-wireless Time-Sensitive Networks that include dynamic wireless (5G/6G) systems. In particular, we consider traffic scheduled according to IEEE 802.1Qbv (time-triggered traffic aka time-aware shaping) and the planning of schedules to cope with dynamic stream sets, dynamic PD distributions, and mobility.

In more detail, we make the following contributions:

- YANG data models to describe PD as enabler for dynamic adaptation to PD. These models can be considered an extension to the existing standard TSN models for static, deterministically bounded delay as defined in [IEEE18-8021Qcc]. The new extended model is sufficiently flexible to describe a spectrum of packet delay definitions, ranging from the classic deterministically bounded PD to histograms of probability distributions derived from runtime PD measurements or PD predictions.
- Concepts for reactive ("break-before-make") and proactive ("make-before-break") adaptation of end-to-end schedules, which are based on the standard NETCONF protocol to either query (poll) dynamic PD distributions by the CNC or use the publish/subscribe paradigm to inform the CNC (YANG Push). Moreover, we show how to integrate PD prediction to proactively trigger the adaptation of schedules. Basing these approaches on YANG and NETCONF facilitates the later integration with existing standard TSN models and mechanisms.
- A set of novel algorithms for planning IEEE 802.1Qbv schedules for dynamic systems including dynamic stream sets, dynamic PD distributions, and mobility. Depending on the algorithm, these algorithms support the maximization of reliability for dynamic PD distributions, fast (re-)planning for dynamic PD distributions and stream sets, and graceful degradation of the streams' timeliness guarantees for worsening channel conditions.
- A new wireless by-pass network architecture with wireless connectivity and optional cloudification of industrial controllers that results in enhanced path redundancy, in improved schedulability, and in additional flexibility to the network operator.

1.4 Relation to Other Work Packages

The approaches presented in this report have been designed in WP3 and have several relations to other work packages and deliverables (see Figure 1):

- WP1: The system concepts presented in this report are based on the use cases and architecture designed in WP1 and presented in deliverable D1.1 *DETERMINISTIC6G Use Cases and Architecture Principles* [DET23-D11] and deliverable D1.2 *First Report on DETERMINISTIC6G Architecture* [DET24-D12], respectively.
- WP2: The approaches for proactive schedule adaptation rely on the prediction of the PD as designed in WP2 and presented in deliverable D2.1 *First Report on 6G Centric Enabler* [DET23-D21]. In general, the stochastic delay characteristics of a mobile network as shown in [MTS+24] have a great impact onto end-to-end scheduling and require new algorithms to calculate robust schedules. The approaches for adapting end-to-end schedules are also related to the orthogonal approach of Packet Delay Correction (PDC) developed in WP2, which aims for reducing the variation of packet delay (PDV). Reduced PDV simplifies the calculation of end-to-end schedules in general and can be combined with the end-to-end scheduling methods discussed in this report, which can deal with PDV.
- WP4: PD distributions are the foundation for calculating robust schedules coping with stochastic PDs. PD distributions have been provided by the latency measurement framework designed in WP4 and described in D4.2 *Latency Measurement Framework* [DET24-D42] and [MNS+23, MSG23]. The validation of robust end-to-end scheduling concepts is performed with the OMNeT++/INET-based network simulator developed in WP4 and described in deliverable D4.1 *DETERMINISTIC6G DetCom Simulator Framework Release 1* [DET23-D41].

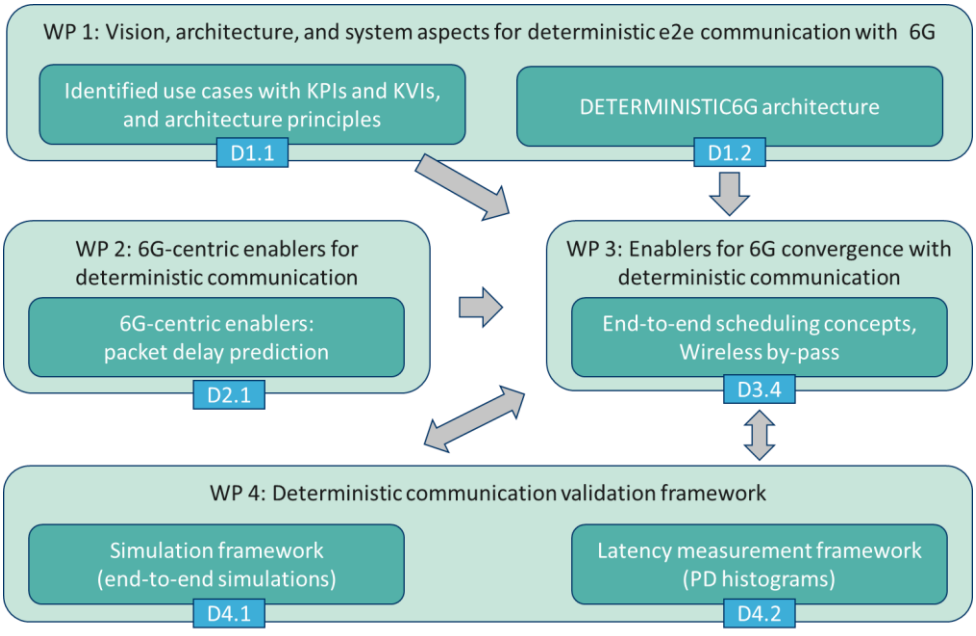

Figure 1: Relation to other work packages

1.5 Structure and Scope of the Document

The rest of this report is structured as follows:

Section 2 focuses on the system aspects of adapting end-to-end scheduling in dynamic systems. The dynamic re-planning of end-to-end schedules by the CNC relies on information from bridges, including the current or predicted PD. To describe this information, we present a YANG data model in this section for stochastic PD as well as different approaches to trigger adaptation reactively or proactively based on NETCONF protocol mechanisms. The complete YANG data model can be found in the Appendix of this report in Section 5.1 and the public project Github[2]. As another system aspect, we discuss implications of the wireless by-pass characteristic to enhance path redundancy and to improve schedulability.

Section 3 focuses on the algorithmic aspects of end-to-end schedule adaptation. We present different algorithms for calculating and adapting time-triggered schedules according to IEEE 802.1Qbv. These algorithms target the different causes of dynamic changes, namely, dynamic stream sets, dynamic PD distributions, and mobility. They provide different features like maximizing reliability with dynamic PD distributions, fast re-scheduling, or smooth handovers of mobile stations with minimal disruption of the schedule.

Section 4 concludes this report with a summary and outlook onto future work.

2 Control Plane Interfaces and Data Models for Dynamic Adaptation

In this section, we present the system aspects that enable the adaptation of end-to-end scheduling in a dynamic system. This includes three major parts:

- Data models to describe the information required for dynamic adaptation. Here, we focus on the stochastic PD distributions of wireless TSN bridges, which are fundamentally different compared to their conventional wired counterparts. PDs of wireless bridges are stochastic in nature with large packet delay variation (PDV). In contrast, existing TSN data models focus on deterministically bounded static PDs. Therefore, we aim for a more flexible approach that incorporates stochastic PDs from fine-grained online measurements. As the modelling language, we use YANG similar to already existing standard by IEEE.
- Approaches to report dynamically changing PD distributions to the CNC to trigger re-planning of end-to-end schedules. Here, we present reactive and proactive approaches based on the standard southbound NETCONF protocol often used between TSN bridges and CNC. To support proactive planning, we also present the integration of PD prediction mechanisms into the framework.
- Utilizing the wireless by-pass characteristic to enhance path redundancy and reduce the number of end-to-end hops by by-passing multiple wired links.

Next, we start by presenting the background and related work. Afterwards, we introduce our system model, before presenting the main contributions of this section, namely, the extended YANG data

[2] https://github.com/DETERMINISTIC6G/deterministic6g_yang_models/

model for PD distribution, reactive and proactive approaches to report dynamic PD distributions, and the wireless by-pass characteristic.

2.1 Background and Related Work

In this sub-section, we briefly describe the technical background and standards required to understand our contributions and serving as a starting point for our extensions. The two most important standard technologies used in this report are: (i) the YANG data modelling language to describe information provided by TSN bridges to the CNC and to configure the bridges by the CNC; (ii) the NETCONF protocol for exchanging information between the bridges and CNC in the network control plane. Therefore, we describe the main concepts of these technologies next. Please note that these brief introductions are not meant as YANG or NETCONF tutorial. They only serve the purpose to keep the document self-contained and readable for the readers not already familiar with YANG or NETCONF.

2.1.1 YANG Data Modelling Language

The YANG data modelling language is an IETF standard defined in [IETF16-RFC7950] to specify how configuration data and state data of network elements – in our case, TSN bridges – is represented and accessed. YANG data models are used by network configuration protocols such as NETCONF (as described in the next sub-section), RESTCONF, or the Constrained Application Protocol (CoAP) to read and write data from respectively to network elements. Due to the hierarchical structure of YANG data models, YANG data model instances can be encoded to other hierarchical representations such as the eXtensible Markup Language (XML) or Java Script Object Notation (JSON). In fact, NETCONF uses XML documents to encode YANG data trees, therefore, we also often use XML documents as examples in this report since most readers might be familiar with XML.

First of all, it is important to distinguish between so-called *configuration data* and *state data*. Configuration data is readable and writeable model data, whereas state data is only readable. The idea is that configuration data can be set by an external network controller to change the configuration of a network element, whereas state data provides state information to network control that cannot be altered by other entities.

YANG comes with several *built-in data types* such as strings, integer numbers with 8, 16, 32, and 64 bits, boolean, enumerations, etc., from which other data types can be derived.

YANG data models are hierarchical consisting of parent-child relationships between data nodes. The `container` keyword is used to define a parent node containing further child nodes in its subtree. A leaf node has no further child nodes but defines a value. An important modelling concept that we later use are *lists* of nodes. List nodes can have unique keys specified by the `key` keyword. A key can also consist of multiple values.

Nodes can also be grouped together using so-called *groupings*. In contrast to a container, a grouping is just a shorthand notation to insert a set of nodes at other places with the `uses` statement. A grouping does not define a parent-child relationship.

The `config` keyword distinguishes configuration data (read/write) from state data (read-only) nodes. Declaring a data node as `config = true` declares configuration data; `config = false` declares a node as state data node.

A node from an existing YANG data model can be extended by further nodes or a complete sub-hierarchy using the `augment` keyword.

Besides defining data models for configuration and state data, YANG data models can also include the specification of *Remote Procedure Calls (RPC)*, i.e., functions to be executed on a remote network element. Closely related to RPCs are so-called *actions*. The difference between RPCs and actions is that actions are connected to certain containers or list data nodes in the tree, i.e., the action is executed on these nodes. NETCONF already defines several operations for retrieving state information or changing the configuration of network elements as discussed in the next sub-section. Since we do not introduce our own RPCs beyond what is already available with NETCONF, we do not further describe the definition of RPCs in YANG data models and explain standard RPCs together with NETCONF below.

Also, *notifications* can be defined with YANG. Similar to actions, notifications can be connected to data nodes (containers, list data nodes).

2.1.2 NETCONF Protocol

The Network Configuration Protocol (NETCONF) is a management protocol to manipulate the configuration of network devices. It is used to retrieve state data or manipulate configuration data of network devices. It goes hand in hand with the YANG standard as already described above: YANG is used to define the data model of state data (read-only) and configuration data (read-write), which is encoded into XML documents to be transported through NETCONF. To this end, NETCONF also defines the operations to make requests to network elements. In terms of the centralized network control paradigm, NETCONF can be considered a southbound protocol between the centralized network controller (CNC in TSN) and bridges to let the network controller control the operation of bridges. In our context, the manipulation of the schedule (gate control list) of TSN bridges supporting scheduled traffic according to IEEE 802.1Qbv [IEEE15-8021Qbv] and retrieving the port-to-port delay of bridges are two important examples of bridge data to be configured or retrieved, respectively.

Figure 2 shows the NETCONF protocol layers as defined in RFC 6241 [IETF11-RFC6241]. The NETCONF client (client for short) connects to the NETCONF server (server for short) via the reliable Transport Control Protocol (TCP). The server shares its configuration data. In our context, the TSN bridge acts as server. Confidentiality, authenticity, and integrity of messages sent between client and server are ensured using, for instance, Transport Layer Security (TLS) [IETF99-RFC2246, IETF11-RFC6101] or the Secure Shell protocol (SSH) [IETF06-RFC4253] running over TCP. NETCONF uses *Remote Procedure Calls (RPC)*, also encoded into XML, for retrieving and editing configuration data by invoking operations on the server side. For instance, the `edit-config` operation is used to change configuration data of the server. Besides request/response communication, NETCONF also supports event-based communication by sending *notifications* from the server to the client. Such event-based communication will show to be useful later in the context of adaptation, to report dynamic network state information from the wireless TSN bridge (6GDetCom node) to the CNC for triggering schedule adaptation.

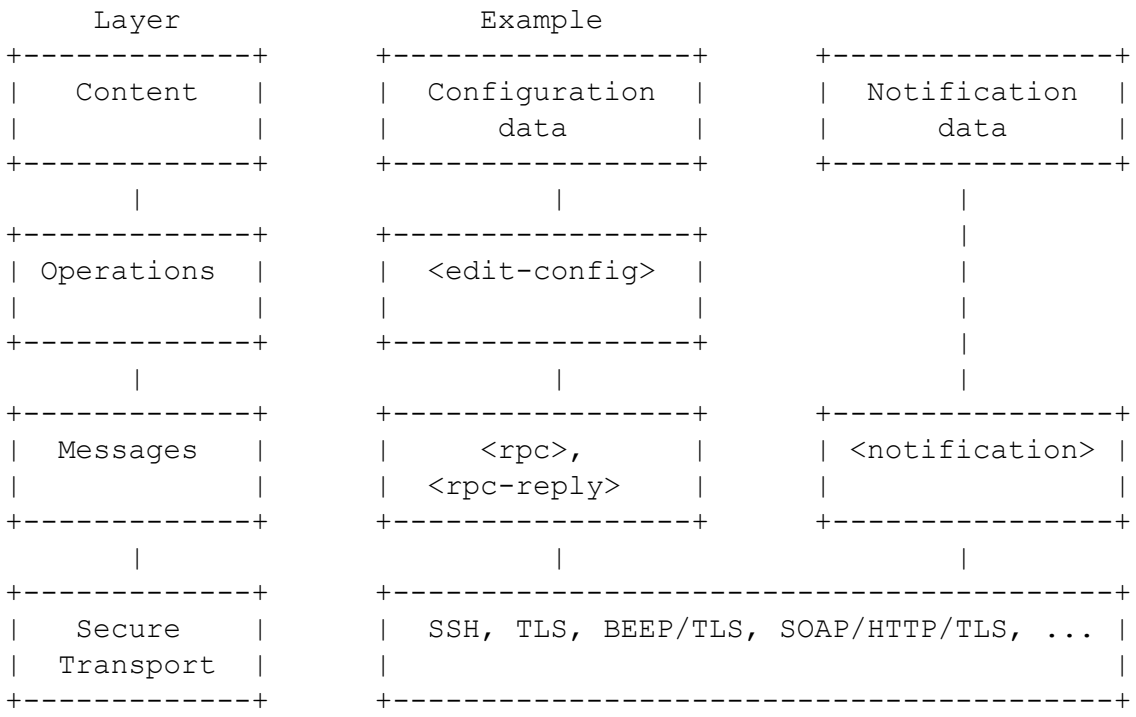

Figure 2: NETCONF Protocol Layers [IETF11-RFC6241]

As soon as the connection between client and server are established, client and server exchange so-called `hello` messages to announce their supported NETCONF version, capabilities (operations beyond the base NETCONF operations), and supported YANG data models. This step is important since NETCONF is an extensible protocol, i.e., different client and server implementations might support different operations and different YANG data models. For instance, a TSN bridge that supports the IEEE 802.1Qbv standard implements a YANG data model to configure the gate control list, whereas a non-TSN bridge would not.

NETCONF defines the concept of *configuration data stores* to hold configuration data that is required for the managed device for its operation. All devices must support a so-called running configuration data store containing the currently operational configuration but might support further configuration data stores. For instance, a candidate data store is often supported to first make changes to the candidate data store without affecting the running configuration, before committing the candidate configuration to the running configuration by invoking the `commit` operation. The candidate configuration can also be locked using the `lock` operation (followed later by an `unlock` operation), which can be useful to avoid inconsistencies when multiple sessions from different clients make changes concurrently to the data store – in general, a single `edit-config` operation is atomic, but with locking and unlocking, multiple `edit-config` operations can be isolated from other sets of concurrent edit-config operations. NETCONF even specifies a so-called `confirmed-commit` operation, which can be useful when updating several managed devices. A `confirmed-commit` first copies the candidate configuration to the running configuration, but rolls-back the changes if no `commit` operation follows within a timeout interval.

Every NETCONF device must support a set of *base operations*, from which we mention a few next to retrieve and change data. First of all, the `get` and `get-config` operations are used to retrieve data from the server. The difference is that `get` can be used to retrieve state and configuration data, whereas `get-config` only retrieves configuration data. Also a filter can be added, for instance, to only retrieve a sub-tree of the data tree as specified by the YANG data model.

Configuration data can be changed with the `edit-config` operation to change parts of the data tree stored in a given configuration data store. The already introduced operations `commit`, `lock`, and `unlock` are used to commit changes to the running configuration, and protect the edited data store from concurrent editing, respectively.

With these operations, we can already retrieve data from a data store and also change data using the request/response type of communication, where the client sends requests (operations via RPC calls) to the server, and the server responds. However, if data is dynamic, such a request/response communication is cumbersome and not very efficient since new data has to be polled by repeated requests to the server. In such a situation, event-based communication is an alternative type of communication, where the server sends notifications to the client instead of waiting for requests from the client. NETCONF also supports event-based communication sending such notifications as shown in Figure 2.

There are several RFCs related to notifications in NETCONF. In the following, we refer to concepts described in RFC 8639 [IETF19-RFC8639], which describes how to subscribe to and receive notifications; RFC 8641 [IETF19-RFC8641], which build on RFC 8639 and describes how to subscribe to updates from a YANG data store (YANG Push); RFC 8641 [IETF19-RFC8641], which in turn builds on the other two RFCs, and describes dynamic subscriptions to YANG events and data stores over NETCONF.

To subscribe to notifications, two types of subscriptions can be distinguished: *dynamic subscriptions* and *configured subscriptions* [IETF19-RFC8639]. For dynamic subscriptions, the subscriber subscribes to notifications using NETCONF RPCs. If the (SSH/TCP) session is terminated, subscriptions established in this session are also automatically terminated. In more detail, the following RPC operations are used to manage dynamic subscriptions, whose meaning is clear from their names: `establish-subscription`, `modify-subscription`, `delete-subscription`, `kill-subscription`. Dynamic subscriptions are a mandatory feature. In contrast, configured subscriptions are an optional feature. Such subscriptions are set up by modifying the configuration of the publisher. They persist across sessions and reboots, as long as the configuration persists. Notifications can also be sent to multiple subscribers.

Further, according to YANG Push, we can distinguish between `periodic subscriptions` and `on-change subscriptions`. Periodic subscriptions send notifications periodically according to some given time interval. Compared to periodically polling information from the data store, they save the request since data is automatically pushed to the subscriber periodically. On-change-subscriptions push updated to the subscriber only when values have actually changed.

Subscriptions also need to define, which parts of a data store are subject to subscriptions to only receive notifications about relevant updates. YANG Push defines two different types of *selection filters*: *sub-tree selection filters* and *XPath selection filters*. A sub-tree filter selects a sub-tree of the data store similar to what can be done with sub-tree filters for get operations as introduced above.

XPath filters use the XPath query language from XML [W3C99-XPath], which defines more powerful concepts to select nodes in an XML tree such as predicates (e.g. sub-string matching), parent-child relationships, etc.

2.1.3 Related Existing YANG Data Models

One goal of this report is to define new YANG data models to model the information required from TSN bridges (in particular, wireless bridges/6GDetCom nodes) to calculate robust end-to-end schedules for time-triggered scheduling (IEEE 802.1Qbv) and to adapt end-to-end schedules to dynamic situations. As we see later in detail, the essential information is the characteristic port-to-port delay of wireless bridges. The standard IEEE 802.1Qcc [IEEE18-8021Qcc] already defines the so-called *bridge delay*, which we introduce next as related work to discuss the differences to our model presented below and motivate the need for a new model.

IEEE Std 802.1Qcc defines bridge delay as the delay that packets experience when passing through the bridge. Individual delays can be defined per (port-pair, traffic class) tuple, where traffic class is a value from 0 to 7 as can be derived from the 3-bit Priority Code Point (PCP) of a VLAN tagged Ethernet frame. The bridge delay explicitly excludes the delay for transmission selection. That is, delay is defined as if queues were empty, the traffic class is permitted to transmit, and the egress port is idle. In other words, queuing delay, which is controlled by scheduling (e.g., the gating mechanism in IEEE 802.1Qbv), is excluded from the bridge delay. These definitions and assumptions are all consistent with our assumptions. In Section 2.3, we also define our so-called port-to-port delay per port-pair and traffic class and explicitly exclude queuing delay since the port-to-port delay should serve as input parameter to calculate robust and adaptive IEEE 802.1Qbv schedules, which essentially controls the queuing delay.

The standard also specifies that delays are defined as worst-case ranges between a minimum delay value to a maximum delay value. That is, delay is deterministically bounded. As becomes clear later, we will extend this notion of deterministic worst-case delay by allowing for the definition of stochastic delay distributions (histograms) to model the characteristic port-to-port delay of wireless bridges.

Moreover, the standard states explicitly that these values (ranges) are not measured, although delays could differ for different configurations of a bridge. Then the delay for the current configuration of the bridge is provided. This is another major difference to our assumptions. We assume that the delay distribution is dynamic and values might change over time significantly. To determine the current delay – or, if delay prediction is used, the future delay – of a wireless bridge, our port-to-port delay is measured and observed online or predicted using data-driven prediction algorithms based on observed data.

The standard further distinguishes between *independent delay* and *dependent delay*. Independent delay is defined as independent of frame length, whereas dependent delay is dependent on frame length. Min/max. values are provided for both, dependent and independent delay. We realize that, in our system, delay may depend on more than just frame length (and traffic class). Therefore, we extend the notion of dependent delay by introducing a generic concept to define dependencies for port-to-port delay. For each dependency, we can report an individual port-to-port delay distribution (histogram).

In summary, our extended models for port-to-port delay embrace the given notion of bridge delay and extend it in different ways to allows for a fine-grained definition of dynamic, stochastic, and dependent port-to-port delay.

2.2 System Model

In this section, we present our system model including the system components and our assumptions, which are the basis for dynamic adaptations. Our system model is shown in Figure 3. We consider networks where *wired IEEE 802 TSN bridges* coexist with *wireless TSN bridges* called *6GDetCom nodes*. This system model follows the fully centralized model of IEEE 802.1Q with a (logically) *centralized network controller (CNC)* in the network control plane controlling the TSN bridges implementing the network data plane, based on a global view onto the network. Bridges expose their state and configuration information to the CNC via the standard *NETCONF* protocol, and the CNC uses NETCONF to change the configuration of bridges. *YANG data models* are used to describe the state and configuration information of bridges. As detailed in the next section, we can utilize NETCONF request/response and pub/sub (YANG Push) mechanisms to update the global view onto the dynamic network state and trigger adaptation of end-to-end schedules.

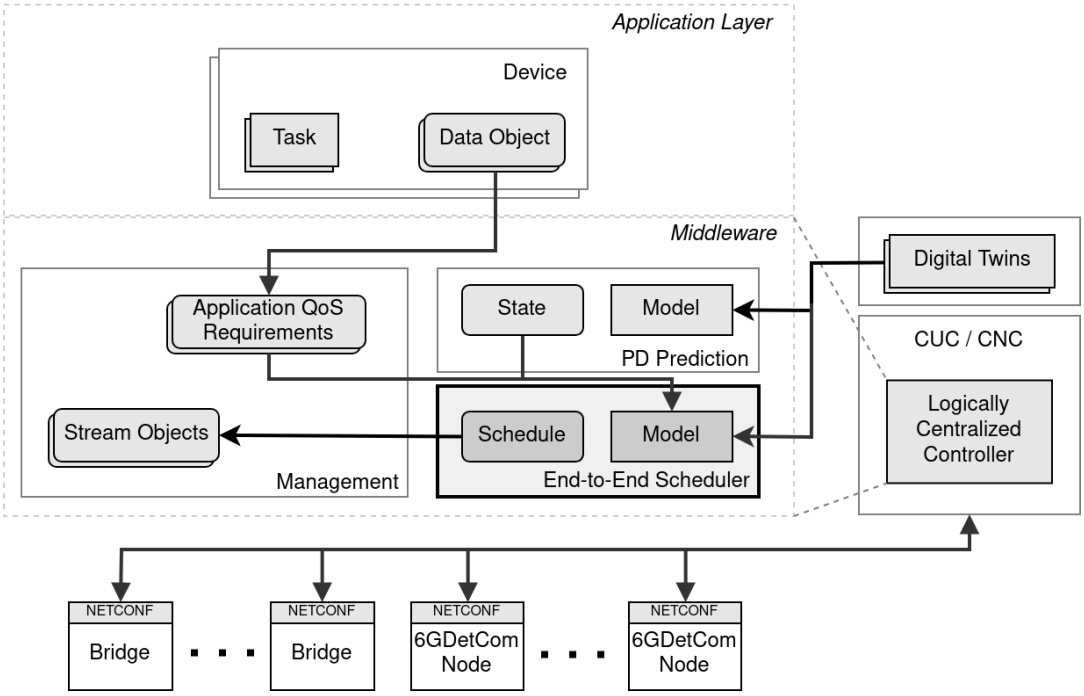

Figure 3: Overview of the system components and their interaction with our end-to-end scheduler design

In particular, we assume that bridges provide information about the *port-to-port delay* between ingress and egress ports to the CNC. The port-to-port delay can either be defined by static worst-case bounds through standard models as defined in [IEEE18-8021Qcc]. In particular, wired bridges or in general bridges whose port-to-port delay has very low variability over time and tight deterministic bounds can use existing standard YANG data models to this end. However, we also consider bridges where (a) the port-to-port delay has large variation following a stochastic distribution, or (b) the stochastic delay distribution is non-stationary with non-constant (dynamic) parameters, where PD measurements and predictions are only valid for a limited time span. Typically, this applies to wireless

bridges due to their characteristic PD. We will later present a YANG data model to describe such stochastic port-to-port delay and NETCONF-based mechanisms to update information of the dynamic delay distributions at the CNC in Section 2.3 and Section 2.5, respectively.

Figure 4 shows more details about the inner structure of a 6GDetCom node with three interfaces in this example. Between the ports of a 6GDetCom node, we have a wireless link between UE and gNB in the 5G system. This leads to a stochastic port-to-port packet delay between the ports of the 6GDetCom node. This port-to-port delay is modelled as histograms as described above for our YANG data model. The port-to-port delay and all other state and configuration data can be accessed through YANG data models via the NETCONF protocol as also already described above. In Section 3, we present algorithms for planning and adapting end-to-end schedules to these dynamic PD distributions.

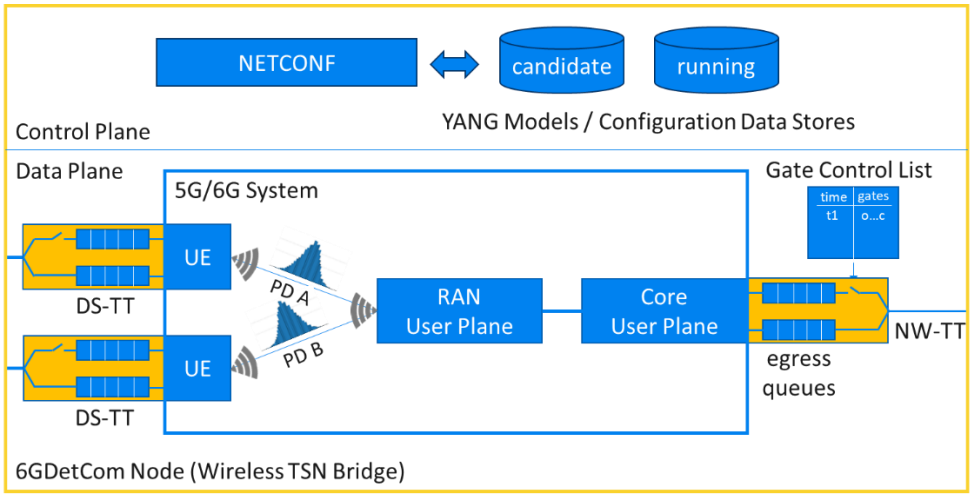

Figure 4: System model of a 6GDetCom Node

Beyond information about the individual dynamic PD distributions of bridges, other context information might be relevant for dynamic adaptation, in particular, to trigger proactive adaptation based on predictions. For instance, information about the mobility of stations can be utilized to predict handovers between 6GDetCom nodes. Also, PD prediction mechanisms might benefit from global information about the network state and environment, e.g. path and speed of stations, number of stations in an area, physical obstacles, etc. As a generic concept to represent all such context information relevant for prediction and sub-sequent dynamic adaptation, we introduce *digital twins (DT)* in our model. By connecting DTs to the CNC, we can capture the context information in the control plane.

In this work, we only consider time-triggered scheduling for scheduled traffic according to IEEE 802.1Qbv [IEEE15-8021Qbv]. To this end, each bridge including 6GDetCom nodes implement the standard transmission gating mechanism, where each egress queue is controlled by a gate. These gates open and close according to a schedule implemented as a timetable, also called *Gate Control List (GCL)*. That is, the GCL defines when which gates are open and closed, and when packets are eligible for transmission. By separating and directing traffic (i.e. TSN streams) to an appropriate egress queue, according to the application requirements for the TSN stream, and then controlling the gates for those queues, the CNC can control the bridge delay range perceived for this bridge for the different

TSN streams[3] including the queuing delay of packets waiting for transmission. The *scheduler* component is responsible for calculating the GCL. We also assume the Per-Stream Filtering and Policing (PSFP) mechanism defined in IEEE 802.1Qci [IEEE17-8021Qci] is used. In more detail, we assume that so-called *stream gates* are applied before the egress queues to filter out packets arriving outside anticipated time intervals to isolate streams from each other and protect from too late or early packets of other streams. It will be the major task of the scheduling mechanisms presented later to calculate and adapt (re-calculate) the schedules for GCLs and stream gate configurations.

To implement proactive adaptation schemes based on predictions for 6GDetCom nodes, we assume a *prediction service* that exposes one or more prediction states to the scheduler. Given a predicted state, the scheduler can query the predicted PD distribution for this state or get notified (i.e., pushed) PD distributions for certain states and then starts pre-calculating schedules with a certain lead time depending on the prediction time horizon. Note that schedule calculation is a complex computational task, and having a lead time to calculate schedules drastically increases the chance to have a schedule ready in time, i.e., before the state has actually changed. Depending on the implementation of the prediction service, it may only expose PD predictions based on the current state. A more sophisticated prediction service may also provide a set of the "most likely" future states. This way, the scheduler can proactively start its computation for each provided prediction state.

The *Centralized User Configuration (CUC)* is used for the interaction between end stations and the CNC. To interact with the application layer, we consider a similar system model as [IEC/IEEE24-60802]. That is, each application specifies data objects that capture the required data exchange between different end devices, along with their reliability and punctuality requirements. The application requirements are forwarded and accumulated by a management middleware, allowing for a global view of the message streams at the CNC. During runtime, the management middleware can notify the scheduler of dynamically changing stream sets, where streams may join or leave the system. From an end-to-end scheduling perspective this means that already configured schedules (GCLs) at bridges need to be re-configured with new schedules including added streams and excluding removed streams. Or in other words: end-to-end scheduling needs to adapt to information about streams that was not available a priori, thus, a static schedule would not suffice in general.

After the scheduler computed an eligible TSN configuration after a significant change of the state requiring new schedules, it notifies the management middleware with the corresponding stream objects. In a subsequent step, these stream objects are used to configure the bridges and the applications by configuring GCLs of bridges and, in case of isochronous traffic with applications synchronized to network time, the schedule of applications for transmitting packets.

An important assumption with respect to station *mobility* is that mobility will be handled transparently and locally within the 5G/6G system in the 6GDetCom node, i.e., end-to-end paths do *not* change when UEs are handed over between base stations (gNB). From a (wireless) TSN bridge perspective, the ports do not change, when the UE moves, neither is the mobile station handed over to another bridge (6GDetCom node). The only effect that possibly becomes visible during a handover is a sudden change of the PD distribution, i.e., the port-to-port delay of the 6GDetCom node. Or in other words: mobility is just another cause for stochastic and dynamic PD distributions, which we anyway already

[3] By applying scheduling for traffic across all bridges of the network accordingly in a coordinated way, the end-to-end latency (minimum and maximum) can be ensured by configuring an appropriate end-to-end schedule.

assume even if stations were completely stationary. Therefore, we do not present approaches for end-to-end scheduling dedicated to mobility but handle it as part of scheduling for stochastic and dynamic PD distributions.

In the following, we first present the YANG data models of 6GDetCom nodes for providing the data required for adapting end-to-end schedules to the scheduler. Afterwards, we describe in more detail the interaction between scheduler and bridges and between scheduler and applications, which is required for adaptation.

2.3 YANG Data Models for Packet Delay

To plan and adapt schedules to dynamic packet delay induced by 6GDetCom nodes, we must be able to model the characteristic port-to-port delay of such wireless bridges first. Then the algorithms for calculating wireless-friendly schedules can use this information as described in the next section to calculate robust schedules and adapt schedules to dynamic port-to-port delay.

To model port-to-port delay, we define a new YANG data model with the following main features:

- Fine-grained modelling of stochastic delay distributions. The granularity of the model can be adjusted and is sufficiently flexible to define probabilistic bounds (probability of delay values within a delay interval) as well as the classic worst-case deterministic bounds (min-max interval with 100 % probability) from IEEE Std 802.1Qcc [IEEE18-8021Qcc].
- Independent and dependent delay distributions: Delay can be defined as independent, i.e., applying to all possible states. We also support a generic notion of dependent delays that are only applicable in certain states. The actual definition of “states” is deliberately kept generic such that PD distributions could depend on any relevant state. For instance, we could define different PD distributions for different frame sizes (frame length dependency), different physical speeds of end stations (mobility dependency), etc. Important for our algorithms for proactive schedule adaptation is mainly that we could calculate schedules for different possible future states proactively, and then switch to the corresponding schedule when the state occurs.
- Validity period: Dynamic packet delay information might only be valid in a certain time frame. Therefore, we add the capability to optionally specify a period of validity.

Here, we only present excerpts from our YANG data model to explain the essential concepts. The full YANG data model can be found in the Appendix of this report in Section 5.1 and in the public project Github[4]

Consistent with IEEE Std 802.1Qcc [IEEE18-8021Qcc], we model port-to-port delay per port-pair/traffic class. Queuing delay (at the egress port) is explicitly excluded from the modelled port-to-port delay. This is also consistent with IEEE Std 802.1Qcc, which defines that bridge delay (as defined by this standard) is provided with zero delay for transmission selection at the egress port. Queuing delay (at the egress port) is controlled by scheduling, for instance, opening and closing gates according to a gate-control list. In contrast, the port-to-port delay modelled here is beyond the control of TSN scheduling. Therefore, the port-to-port delay is provided as if gates were open. The port-to-port delay

[4] https://github.com/DETERMINISTIC6G/deterministic6g_yang_models/

depends on the bridge characteristics for transferring packets from the ingress port to the egress port. In a 6GDetCom node this includes the 6G wireless transmission (see e.g. [DET23-D21]).

The basic idea to flexibly model port-to-port delay is to use histograms as shown in Listing 1 in the `histogram` grouping of the YANG data module. The histogram consists of a number of bins as defined by `bin-count`. Bins can have different widths. If the first bin does not start at delay value zero, then the node `start` can be used to define the start of the first bin. The interval before this start value have probability zero. The node `count` defines the number of values within a bin. Note that `count` can also be translated to a relative frequency very easily if `count` is interpreted as numerator and the sum of all counts as denominator of the relative frequency. Histograms with bins and counters are also used for performance measurements for 5G network functions [3GPP24-28552], in particular, for delay measurements. If PD delay distributions are available already as continues probability density functions, for instance, as a result of delay predictions rather than measurements, they could be discretized to provide a histogram. The node `tail` can be used optionally to define how many values are in the tail of the distribution after the last given bin, which effectively defines a final bin reaching to infinity (unbounded delay distribution).

```
grouping delay-histogram {
    description
        "Delay histogram";
    leaf start {
        type uint64;
        description
            "The start value of the first bin in nano-seconds.
            If not specified, the first bin starts at 0.";
    }
    leaf bin-count {
        type uint32;
        mandatory true;
        description "Number of bins.";
    }
    list bin {
        description "Bins of histogram.";
        key index;
        leaf index {
            type uint32;
            mandatory true;
            description "The index of this bin.";
        }
        leaf width {
            type uint64;
            mandatory true;
            description
                "The width of this bin in nano-seconds.";
        }
        leaf count {
            type uint32;
            mandatory true;
            description "Count of values in this bin.";
        }
    }

    leaf tail {
        type uint32;
        description
            "Count of values in the tail of the histogram
            after the upper bound of last bin until infinity.
            Can be used to define an unbounded distribution.";
    }
}
```

Listing 1: YANG data model – histogram

This definition of histograms shows to be quite flexible when modelling port-to-port delay as demonstrated next. First, we can model the classic deterministic definition of delay as specified in IEEE Std 802.1Qcc [IEEE18-8021Qcc] as worst-case min/max bounds as shown in Listing 2 with a single

bin, assuming a minimum delay bound of 1 ms and maximum delay bound of 10 ms in this example. The count value can be set to any value greater than zero here since the total count of all bins summed up corresponds to 100 % probability in a histogram.

```
<bin-count>1</bin-count>
<!-- first bin starts at 1 ms -->
<start>1000000</start>
<bin>
    <index>0</index>
    <!-- 10 ms bin width-->
    <width>10000000</width>
    <count>1</count>
</bin>
```

Listing 2: Deterministic delay bounds

A histogram with more bins is shown in Figure 5. This histogram is bounded on the right hand-side, i.e., 100 % of the values fall into the given bins. By adding a `tail` node, we could extend this distribution ad infinitum, i.e., less than 100 % of values are in the interval from zero up to the upper bound of the last specified bin, and the tail contains the rest of the values (unbounded distribution).

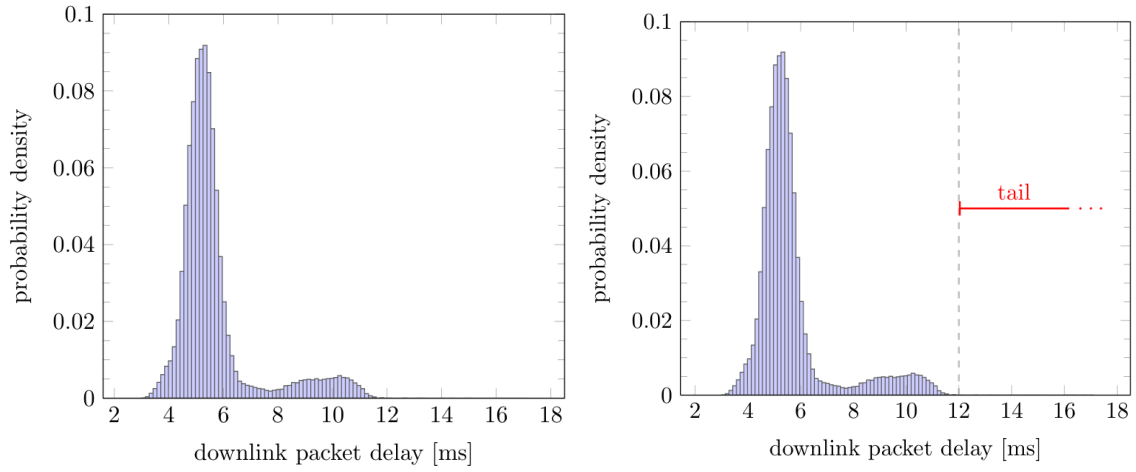

Figure 5: Histograms for bounded (left) and unbounded (right) packet delays

Next, we assign one or several histograms to a (ingress-port, egress-port, traffic-class, index) tuple as shown in Listing 3 by augmenting the standard bridge component node from the YANG module dot1q-bridge.yang [IEC/IEEE24-60802]. This is also consistent with IEC/IEEE 60802, where dependent and independent delays are added below the bridge component node. The node `port-to-port-delays` contains a list of `port-to-port-delay` nodes indexed by the combined key (ingress-port, egress-port, traffic-class, index), which are all based on standard IEEE types. For instance, the traffic class is a number from 0 to 7, corresponding to the Priority Code Point (PCP) header field in the VLAN tag. Each `port-to-port-delay` node defines a histogram by using the `delay-histogram` grouping presented above. Using the `index` node, we can assign multiple histograms per ingress-port, egress-port, traffic-class.

```
augment "/dot1q-bridge:bridges/dot1q-bridge:bridge/dot1q-
bridge:component" {
    container port-to-port-delays {
        config false;
        list port-to-port-delay {
        key "ingress-port egress-port traffic-class index";
        leaf ingress-port {
            type dot1qtypes:port-number-type;
            config false;
            mandatory true;
            description "Unique number of ingress port.";
        }
        leaf egress-port {
            type dot1qtypes:port-number-type;
            config false;
            mandatory true;
            description "Unique number of egress port.";
        }
        leaf traffic-class {
            type dot1qtypes:traffic-class-type;
            config false;
            mandatory true;
            description "Traffic class (0..7)";
        }

        leaf index {
            type uint16;
            config false;
            mandatory true;
            description
                "Index to define multiple histograms per port-pair
                and traffic class.";
        }
        uses delay-histogram;
    }
    container validity-period {
        container valid-from {
            description
                "Given delays are only valid at or after this
                point in time, specified as PTP timestamp.";
            uses ieee802:ptp-time-grouping;
        }
        container valid-until {
            description
                "Given delays are only valid until this
                point in time, specified as PTP timestamp.";
            uses ieee802:ptp-time-grouping;
        }
    }
```

```
        leaf dependency-class {
            type enumeration {
                enum "independent";
                enum "dependent";
            }
            description
                "Are the given delays only applicable under certain
                conditions (e.g., for frames of certain length)?";
            }

}
```

Listing 3: Histograms assigned to (ingress-port, egress-port, traffic-class)

Multiple histograms can be specified per port-pair/traffic-class to define independent and possibly multiple dependent port-to-port delay distributions as indicated by the node `dependency-class`. For instance, we can define an independent delay distribution and a frame-length-dependent delay distribution for the same port-pair/traffic-class combination. Beyond length dependencies, our YANG data model is generic and extensible to define arbitrary further dependencies in the future by defining custom `dependencies` as shown in Listing 4. Thus, depending on the state, individual delay distributions can be defined and communicated to the CNC to calculated state-dependent schedules. In particular, this facilitates proactive adaptation schemes by proactively calculating schedules for different (possible) future states and then use the schedule which matches the actual state.

```
augment "/dot1q-bridge:bridges/dot1q-bridge:bridge/dot1q-
bridge:component/port-to-port-delays" {
    when "dependency-class = 'dependent'";
    container dependencies {
        container length-dependency {
            leaf min-frame-length {
                type uint32;
                description
                    "Values apply only to frames equal or greater
                    than this value.";
            }
            leaf max-frame-length {
                type uint32;
                description
                    "Values apply only to frames equal or smaller
                    than this value.";
            }
        }
    }
}
```

Listing 4: Extensible dependency model

Finally, the node `validity-period` defines the time frame in which the given distribution is valid. Such a validity period is especially useful for dynamic systems where the delay distribution is expected to change (instead of the static distributions as defined currently in the IEEE standard). This facilitates querying new distributions latest when the old distributions become invalid to adapt the schedule or also in advance. If latency prediction concepts are used, one could even define a validity period starting in the future. For instance, if a mobile device is predicted to move around a corner in 5 s from the time of querying the delay distribution, the distribution could be marked as valid only in 5 s, when the delay distributions is expected to change.

In summary, our YANG data model defines the schema shown as schema tree in Listing 5.

```
module: port-to-port-delay

  augment /dot1q-bridge:bridges/dot1q-bridge:bridge/dot1q-
bridge:component:
    +--ro port-to-port-delays
       +--ro port-to-port-delay*
       |     [ingress-port egress-port traffic-class index]
       |  +--ro ingress-port     dot1qtypes:port-number-type
       |  +--ro egress-port      dot1qtypes:port-number-type
       |  +--ro traffic-class    dot1qtypes:traffic-class-type
       |  +--ro index            uint16
       |  +--ro start?           uint64
       |  +--ro bin-count        uint32
       |  +--ro bin* [index]
       |  |  +--ro index    uint32
       |  |  +--ro width    uint64
       |  |  +--ro count    uint32
       |  +--ro tail?            uint32
       +--ro dependency-class?     Enumeration
       +--ro validity-period
       |  +--ro valid-from
       |  |  +--ro seconds?       uint64
       |  |  +--ro nanoseconds?   uint32
       |  +--ro valid-until
       |     +--ro seconds?       uint64
       |     +--ro nanoseconds?   uint32
       +--ro dependencies
          +--ro length-dependency
             +--ro min-frame-length?   uint32
             +--ro max-frame-length?   uint32
```

Listing 5: Schema tree of port-to-port delay YANG data model

2.4 Interaction between Scheduler and Application

In conventional settings, a scheduler has to be able to adapt TSN configurations to joining and leaving streams, i.e., adapt the schedule to dynamic stream sets. Moreover, the specific properties of our wireless system including, in particular, dynamic PD distributions might mandate further interaction between scheduler and application to deal with degrading system performance. In the following, we discuss coordination steps between scheduler and applications.

2.4.1 Incremental Scheduling

While the schedule can easily be adapted for leaving streams, joining streams may affect the ones that are already scheduled. As a consequence, a TSN scheduler middleware must perform an acceptance test to ensure that no reliability or punctuality guarantee is impaired by admitting additional streams.

From the perspective of the TSN scheduler, the converged architecture of wireless and wireline communication offers novel possibilities for realizing such acceptance tests. On the more traditional side, joining streams can be accepted *unconditionally*; that is, by accepting a joining stream, the scheduler guarantees the application that its stream requirements are met under any circumstance with respect to the specified reliability. In comparison, a holistic design may *conditionally* accept joining streams, where the scheduler only provides reliability and punctuality guarantees for certain network and environmental conditions.

To illustrate the impact of this design choice, we consider a simple realization of a conditional acceptance test in the following: On accepting a joining stream f, the TSN scheduler guarantees that it can meet f's end-to-end latency guarantees with its required reliability *as long as the TSN scheduler can guarantee the same for all streams with higher priorities.* As a consequence, the scheduling middleware reserves the right to drop f at a later point in time, e.g., if

- another joining stream with higher priority requires additional resources, or if
- the wireless channel quality degrades to a point where the schedule has to perform triage.

2.4.2 Reaction to Degrading Application Performance

Whereas dropping f in the above cases can be quite drastic, an acceptance test may also specify a graceful degradation of f's punctuality guarantees. Depending on the application, QoS degradations often appear more acceptable than arriving at a complete system halt. The following provides examples of possible tradeoffs that our scheduler design can offer to the application layer.

On the one extreme, our scheduler can aim to uphold the streams' reliability guarantees at the cost of introducing additional tardiness. For instance, when the scheduler can no longer guarantee that the stream f is delivered within $10\ ms$ and a reliability of $99.99\ \%$, it may prolong f's acceptable end-to-end latency to $15\ ms$ (assuming that this latency can be provided by $99.99\ \%$). On the other extreme, our scheduler can enforce the streams' punctuality requirements at the cost of impairing their reliability guarantees. For instance, f's end-to-end latency stays at $10\ ms$ but its reliability degrades to below $99\ \%$. As both extremes can result in unbalanced QoS guarantees, a hybrid combination is expected to result in better tradeoffs.

2.5 Interaction in Control Plane to Adapt to Dynamic Packet Delay

Next, we consider the adaptation of end-to-end schedules to dynamic PD distributions, i.e., a change of the stochastic properties such as a shifted mean delay or changing PDV. This involves three major steps:

1. Retrieving information about the (dynamic) PD distribution from the bridges or from the prediction service by the scheduler through the CNC.
2. Calculating a new end-to-end schedule (GCLs for each bridge) by the scheduler.
3. Pushing the new configuration (GCLs) to the bridges.

For Step 1 and Step 3, we describe how to use standard NETCONF mechanisms and also discuss the implications for the scheduler. The YANG data models for describing PD communicated in Step 1 over NETCONF have already been presented in Section 2.3.

Step 2 is the algorithmic part of adaptation, which is covered in detail in Section 3.

We distinguish between two major approaches:

- *Reactive adaptation ("break-before-make")*: The scheduler calculates ("makes") a new schedule only after it is informed that the PD has actually changed. Since schedule calculation takes significant time, this means that the old schedule stays effective until a new schedule has been calculated, although the old schedule has not been calculated for the current situation (PD distribution). Consequently, until the new schedule is available, given guarantees (latency bounds and deadlines) might be violated ("break"). This problem is critical since schedule calculation is a complex computational task, and therefore, the transition period might be relatively long (several seconds at least even with optimized algorithms for calculating schedules).
- Proactive adaptation ("make-before-break"): To tackle the obvious problem of reactive adaptation, proactive adaptation follows a "make-before-break" approach where schedules are calculated ahead of the time when they are actually needed and become effective. This obviously requires some PD prediction mechanisms to define when and how PD distributions change significantly in their characteristics like mean, variance, etc., which is a complex task on its own out of the scope of this report. PD prediction is considered in the DETERMINISTIC6G project [MNS+23] and it is subject to future work how to further enhance the prediction of PD for future states. In this report, we only provide an overview of PD prediction and focus on the implications for scheduling.

We start by presenting the mechanisms for reactive schedule adaptation, before we consider proactive schedule adaptation.

2.5.1 Reactive Schedule Adaptation

Periodic Polling

The straightforward approach to implement reactive adaptation is periodic polling of all bridges that are providing dynamic port-to-port delay information for the current distribution of the port-to-port delay (histogram as defined in Section 2.3). Bridges providing only static information on port-to-port delay such as static worst-case bounds obviously only need to be polled once since these values are never updated. Whether a bridge can provide dynamic port-to-port delay information would be

advertised through capabilities in NETCONF, so the CNC can selectively poll bridges periodically that provide dynamic information.

Figure 6 shows a periodic polling sequence with a given polling interval (we only show the sequence of actions here only for one bridge, but the same actions would be executed for each bridge providing dynamic information). With NETCONF and YANG, this can be implemented by calling the `get` operation on the `port-to-port-delays` node in the YANG data model defined above. After calculating the schedule, the bridges can be configured using the `edit-config` operation followed by a commit operation to configure the new gate parameters at the bridge including the so-called `admin-control-list`, which is the new configured GCL. The new admin control list becomes operational at the start of the next cycle (`admin-base-time` + n*`admin-cycle-time` for the smallest integer n such that the time is in the future).

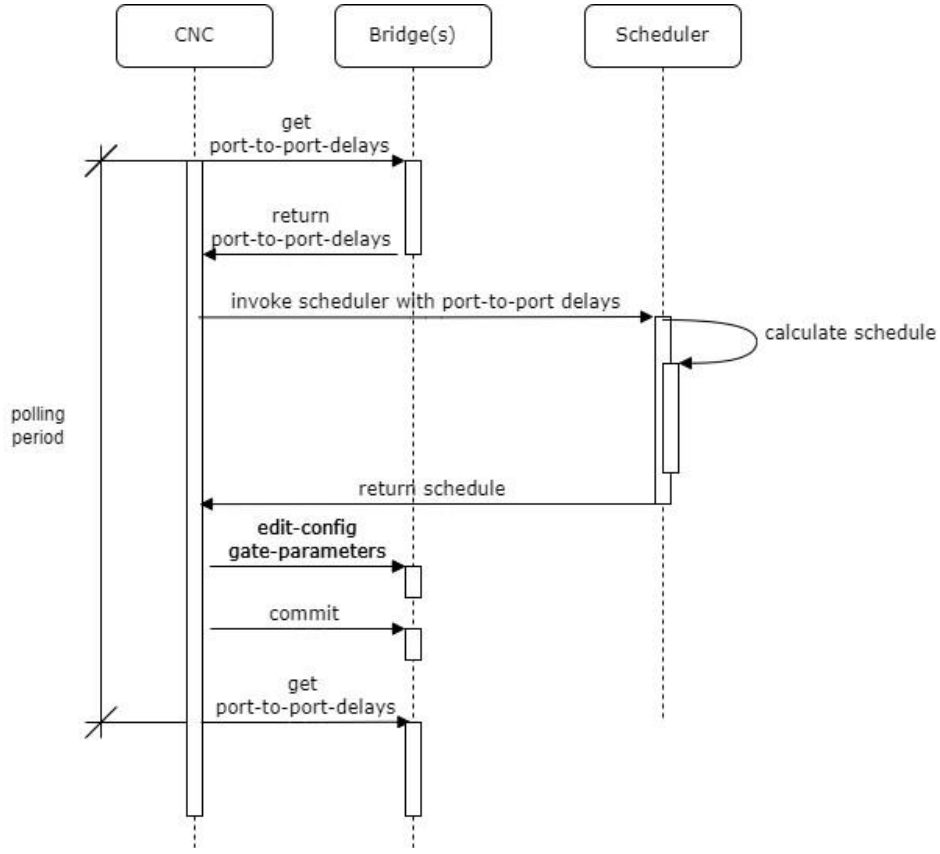

Figure 6: Sequence diagram – Periodic polling of PD and updating of bridges

There are several problems and difficulties with this approach. The first problem is the time, say t_{calc}, that it takes to calculate the schedule, which is typically much longer than the rest of the interaction and can easily be several seconds even with fast heuristics to calculate schedules (see Section 3). If a new schedule is required – i.e., if the PD distribution changes significantly and polling starts immediately when the PD distribution changes –, then it will take at least t_{calc} to install a new schedule, i.e., the schedule is at least broken for period t_{calc} in this case – we call this the *broken period* in the following.

However, this is not even the worst case (longest duration) of the broken period. An old schedule might already become outdated without being noticed by the CNC *before* a new delay distribution is polled, i.e., during the current polling period before the next poll request, which further prolongs the broken period. In fact, the last polled PD distribution might already become outdated immediately after it is polled, and the new schedule is already broken, when it is installed. Then it takes another full polling period plus t_{calc} before a new schedule will be installed. And then the same situation might repeat again, and schedule adaptation lags behind forever with only broken schedules installed in the worst case.

Clearly, if PD distributions can change *any* time, and t_{calc} is greater than zero, no update scheme can ever guarantee a lower bound on the broken period. However, polling introduces an artificial "dead time" equal to the polling interval plus t_{calc} before it can effectively react. So, one question is: Can we improve on the lower bounds of the broken period with another scheme? We will come back to this question later after having completely discussed the polling scheme.

Other problems of the polling scheme are the selection of polling period and the induced overhead, which we discuss together next. Obviously, a long polling period leads to long broken time bounds but low communication overhead and load onto bridges, which need to process and respond to poll requests. A short polling interval leads to faster reaction times and high communication and processing overhead. A reasonable lower bound for the polling period would be the time to calculate a new schedule since it is not efficient to query for new delay values as long as the old values have not been processed into a new schedule. Using YANG Push, we can use periodic subscription as introduced above to let bridges report their current port-to-port delay periodically. If notifications from all bridges have been received (one round of updates is finished), the scheduler can be invoked. This saves the periodic get-requests from Figure 6; the response message become notifications as shown in Figure 7. Besides this difference, the fundamental problems remain: How to reduce the influence of the polling period onto the broken period? How to choose the notification interval? These questions directly lead to an event-based communication approach, not based on periodic polling or periodic notifications, but based on actual changes of delay distributions, i.e., based on the delay characteristics, as discussed next.

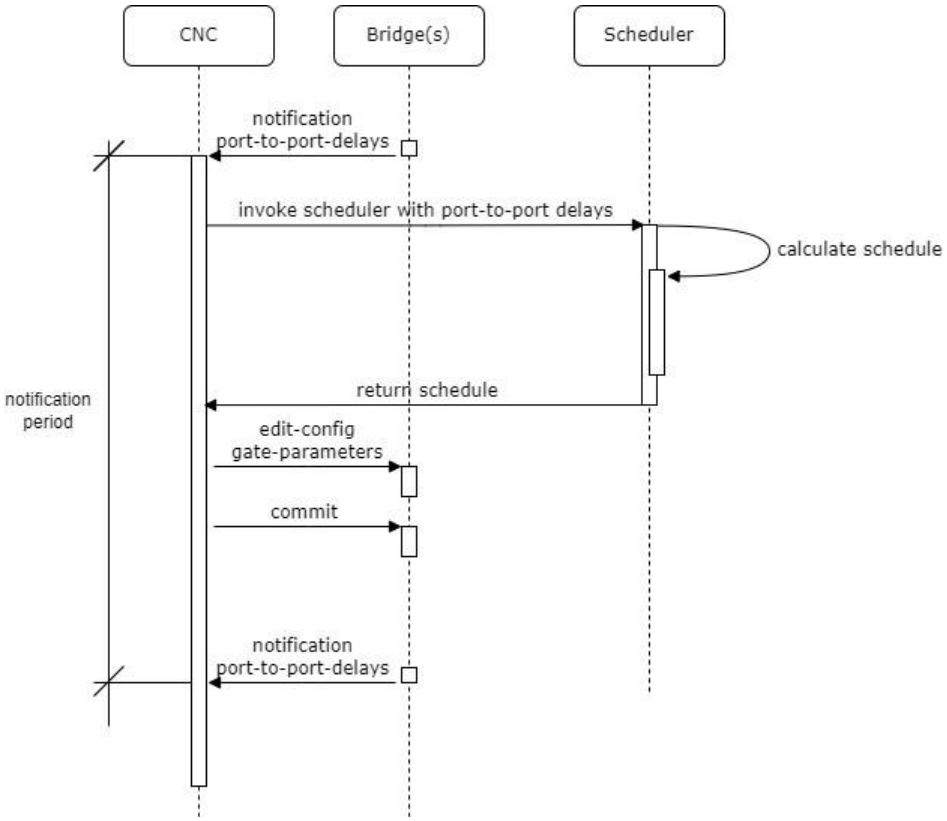

Figure 7: Sequence diagram – Periodic subscriptions for reporting dynamic PD distributions and scheduling. For the sake of simplicity, the calls to establish subscriptions (operation `establish-subscription`) have been left out

Event Notifications

To eliminate the dead time caused by the polling period of the polling approach above, we now consider an event-based approach based on value changes (on-change subscriptions in YANG Push) rather than periodic updates. Whenever the port-to-port delay changes significantly, the scheduler is notified via the CNC as fast as possible. Figure 8 shows the sequence of actions of this approach. Although it looks similar to the periodic subscriptions above, the major difference is the missing notification period, which is replaced by a check for delay value changes at the bridge.

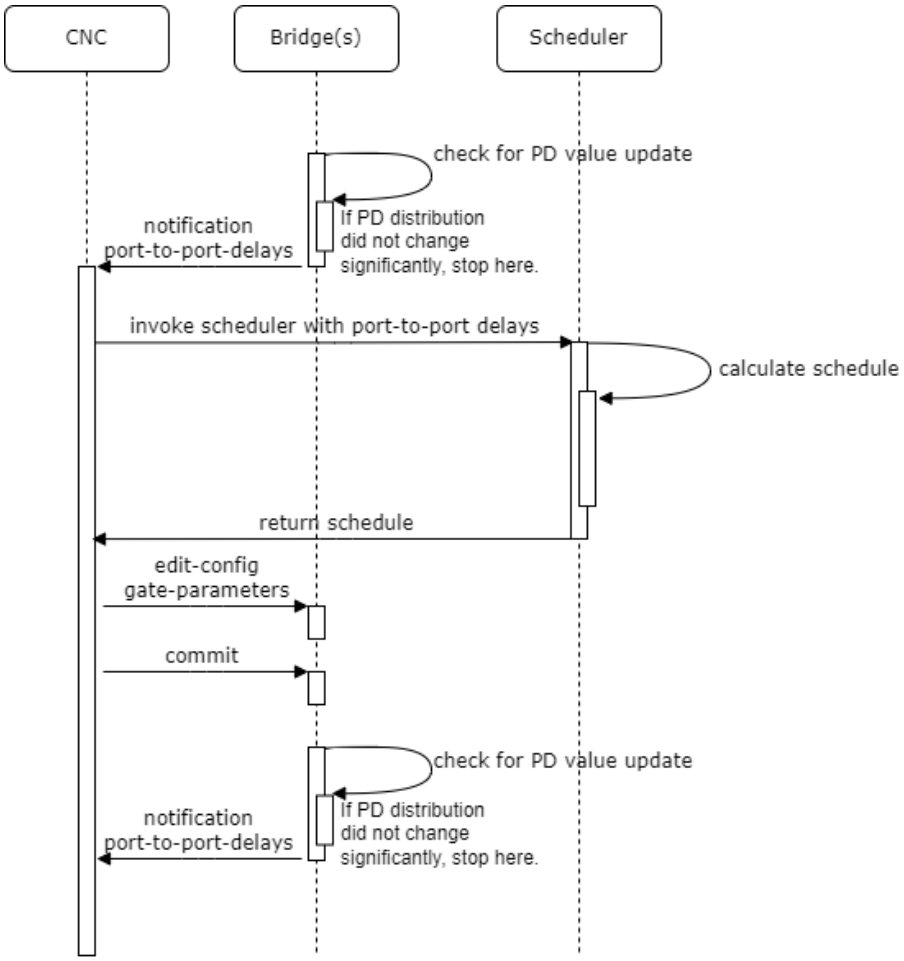

Figure 8: Sequence diagram – Event-based updates of PD and schedule adaptation.
For the sake of simplicity, the calls to establish subscriptions (operation `establish-subscription`) have been left out

The upper bound of the broken time (time to react to a significant change of PD requiring a new schedule), is now dominated by the time t_{calc} required to calculate a new schedule, plus a minor delay to trigger and communicate the updated port-to-port delay to the CNC and push new schedules to the bridges.

Since polling is not used anymore, also the problem of defining a suitable polling interval is solved. Instead, a new problem arises: Which delay *value updates* should be provided to the CNC? This question is directly related to the question, when is a change of PD significant to require a new schedule? A naive answer to the first question could be to simply send all changes to the CNC (send all updates, let the CNC sort it out). On the one hand, this does potentially induce big overhead, which could be saved by using YANG Push features. As mentioned above, XPath selection filters can be used on subscriptions to define filters on the YANG data model. XPath includes the possibility to define predicates on values, such as *value > some other value*. This would allow for what is also known in Publish/Subscribe as content-based filtering that the scheduler uses to specify thresholds on value changes that are considered significant for scheduling, and suppress insignificant value updates.

On the other hand, it does not answer the second question: Which updates are actually significant such that a new schedule needs to be calculated? After all, the schedule calculation implies big processing overhead, even if executed on a fast edge cloud server infrastructure hosting the CNC. To answer this question, some knowledge of the robust scheduling algorithms is required, which can cope with stochastic PD. These algorithms are described in Section 3 but in a nutshell, we can summarize the requirements as follows: For the scheduler it is important to know the probability of the frame transmissions completing within some time interval $[d^{min},d^{max}]$ allocated by the schedule for the transmission. For adapting the schedule, e.g., by increasing or decreasing the size of allocated intervals, it is important to know whether the schedule actually falls behind the desired reliability or exceeds the desired reliability. Notifications towards the CNC and scheduler are only sent if the actual reliability — i.e., the 5G reliability guarantee when keeping $[d^{min},d^{max}]$ — significantly differs from the 5Greliability required for upholding the streams' guarantees end-to-end.

Content-based subscriptions (informing bridges which updates are relevant) also would give the possibility for more sophisticated, hierarchical adaptation strategies. To meet reliability requirements, one could either adapt the (radio) resources allocated to streams at the 6GDetCom node, i.e., solve the problem by adapting the 5G system. Or one could adapt the resources allocated by the end-to-end schedule (allocated time intervals) to solve the problem "end-to-end" on the TSN level. Informing the 6GDetCom node of what changes are considered significant from the end-to-end scheduling perspective allows the 6GDetCom node to make an informed decision whether it should solve the problem locally in the 5G system, or escalating the problem to TSN network control to solve the problem end-to-end. Obviously, such a hierarchical adaptation approach is challenging. Since this report is focused on the adaptation of end-to-end schedules, we consider hierarchical adaptation to be out of the scope of this report and part of future work. But we conclude that using content-based subscriptions could reduce overhead by only reporting significant changes to the scheduler and support hierarchical adaptation approaches by enabling the 6GDetCom node to make informed decisions whether to adapt locally.

2.5.2 Proactive Schedule Adaptation

If we want to further reduce the lead time of scheduling below t_{calc}, we cannot rely on a reactive scheduling approach that only starts calculating schedules when the PD has already changed. Instead, we must calculate schedules proactively to ideally have them ready when the PD actually changes. This is only possible if we employ PD prediction to predict the future packet delay and use these predictions as input to calculate end-to-end schedules. If predictions are sufficiently accurate to predict the PD distribution that will occur t_{calc} time units in the future, we could theoretically reduce the broken time to zero by always having a correct schedule ready in time. So the duration t_{calc} to calculate a new schedule is still critical here. If t_{calc} is long, then also the PD prediction must be able to predict PD distributions for a long time in the future to fully compensate for the long calculation time.

A simple protocol is depicted in Figure 9. Note that we replaced bridges here by the prediction service since the prediction algorithms typically require substantial resources, which we might not associate with a classic TSN bridge but some service running, for instance, in an edge cloud infrastructure. Moreover, the prediction service might not only consider local data of a bridge to make predictions but could utilize *all* available *global* context information, including information from the network and the environment (e.g. station mobility, obstacles, etc.).

We augment port-to-port delays provided as YANG data models by a validity period as defined above in our YANG data model. In particular, the attribute `valid-to` specifies that the current port-to-port delay distribution will expire at this point in time. Then the CNC can query for a new distribution t_{calc} time units in advance and push the new schedule to the bridges before the expiration time. Bridges will use the new schedule at this time or earlier, if prediction indicates that the new distribution is already valid before the old schedule expires using the model attribute `valid-from`. TSN supports to make the configured `admin-control-list` (new GCL) operational at the `admin-base-time` if the `admin-base-time` is in the future, i.e., candidate configuration data store of bridges can already be configured in advance and automatically make the new schedule operational.

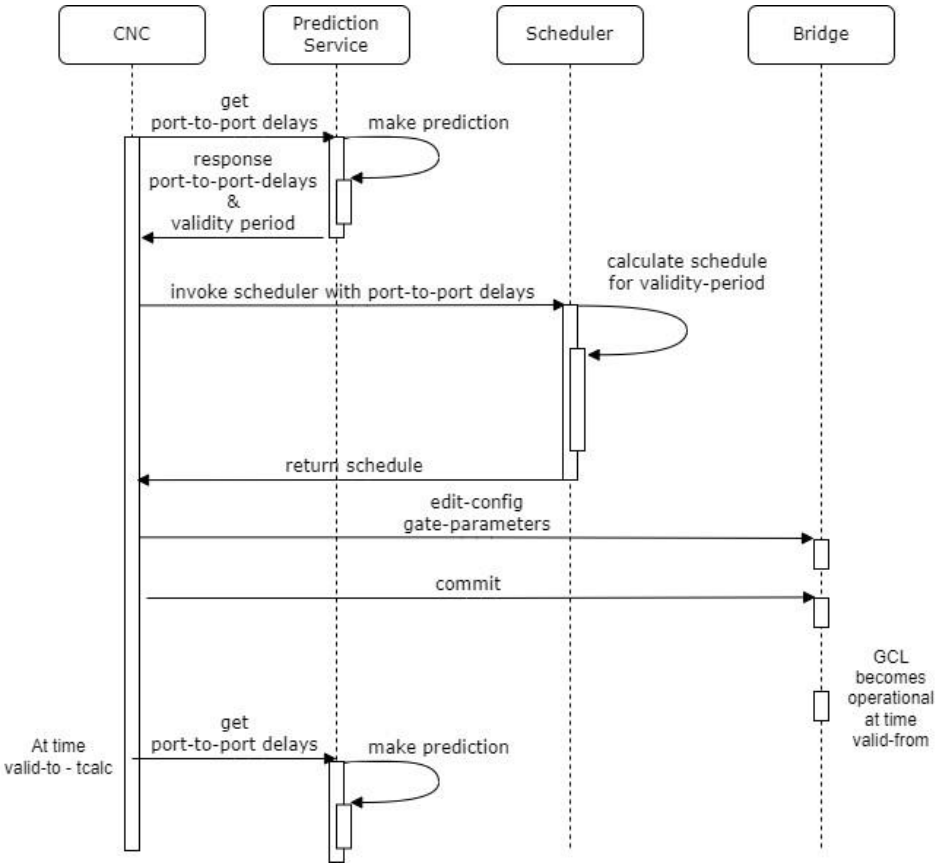

Figure 9: Sequence diagram – Proactive adaptation of schedules

Such a predictive approach can be made more sophisticated depending on the capabilities of the prediction mechanism. Although a detailed discussion of prediction mechanisms is out of the scope of this report and discussed elsewhere in the DETERMINSTIC6G project in dedicated reports and publications [MNS+23, MSG23], we provide here a short description of the prediction concepts and then discuss implications for the adaptation of end-to-end scheduling.

Overview of Data-driven Prediction

The accurate prediction of PD characteristics is essential for achieving dependable time-critical communications in 6G networks. With the evolution of various network and traffic conditions in 5G/6G systems, the ability to estimate the probability density function (PDF) of packet delay is required, e.g., to calculate robust end-to-end schedules. Data-driven approaches, particularly those

leveraging ML techniques, have been proposed to obtain the relationship between packet delay and various conditions based on real-world measurement data. Specifically, conditional density estimation (CDE) of delay involves creating a mapping (h_ω) from the conditions X (e.g., the network state) to the parameters θ (e.g., mean, skewness, variance, long tail) of the density function, i.e., $\theta = h_\omega(X)$. Deliverable D2.1 "6G Centric Enablers" [DET23-D21] provides a detailed description of the proposed approaches to solve the CDE problem. In essence, the goal of such data-driven delay predictors is to obtain a delay characterization in the form of a PDF given a specific network and traffic state.

Given the dynamic nature of 5G-Adv/6G systems concerning network and traffic, it is not sufficient to produce delay predictions once and use that to calculate end-to-end schedules. The delay predictions of 5G made at a certain time are expected to change over time as the system state evolves stochastically. Therefore, it is useful to provide a quantitative measure of confidence in delay predictions. Generally, this confidence should decrease with the forecast lead time, meaning the further into the future the prediction, the lower the confidence in the estimated delay PDF. The confidence level can be defined as the difference between the estimated delay PDF (ranging over a few network conditions) and the marginal delay PDF (ranging over many/all network conditions) [Del04]. Conversely, the "time of predictability limit" (L_{pred}) can be defined as the time when it is no longer possible to distinguish between the estimated and marginal PDFs, indicating that predictability diminishes. From an end-to-end scheduling perspective, on one hand, a larger L_{pred} implies a longer duration over which the end-to-end schedule is expected to meet the delay-reliability requirement. On the other hand, if L_{pred} is smaller, faster computation and adaptation of schedules are required.

Implications for predictive end-to-end scheduling

A reactive adaptation mechanism would solely consider the current packet delay PDF, e.g., as approximated by its parameters $\theta = h_\omega(X)$. However, in case L_{pred} is relatively small, the end-to-end scheduler may not be able to find eligible solutions within the required timespan. As a consequence, the end-to-end scheduler may have to fall back to a (potentially more conservative) precomputed TSN schedule.

Our end-to-end scheduler design supports proactive approaches by incorporating the following:

- Multiple port-to-port delay histograms can be specified with our YANG data models of Section 2.3. Each histogram can be associated with dependencies that capture the current network condition X, defining under which circumstances the provided port-to-port delay is applicable. The notion of network conditions is deliberately generic to be able to cover a broad range of dependencies.
- By exposing a set of the most likely packet delay histograms, i.e., $\{(X_1, \theta_1), \dots (X_k, \theta_k)\}$, our end-to-end scheduler can proactively start finding eligible schedules S_i for each network condition X_i. Hence, if the prediction service notifies our scheduler of changing network conditions thereafter, we can immediately deploy the precomputed schedule S_i without additional overhead.

Further details on possible realizations of proactive end-to-end scheduler are provided in Section 3.4.

2.6 Network Topology with Wireless By-Pass

Next, we focus on another systems aspect that impacts end-to-end scheduling: leveraging the wireless network to by-pass a chain of wired links. Such a wireless by-pass has positive effects onto the complex task of end-to-end scheduling since the simplified topology has fewer hops on end-to-end paths. A simplification of the complex scheduling task is particularly beneficial if schedules need to be adapted to dynamic changes such as applications and their streams joining of leaving the system. Thus, the wireless by-pass also simplifies such adaptations and reduces the time to adapt, e.g., to calculate new schedules.

We first introduce the architecture of the wireless by-pass before discussing its impact.

2.6.1 The Wireless By-pass

Figure 10 shows an example of a wired TSN network and its two main components: (1) the "End station" component, defined in IEEE Std 802-2014 [IEEE14-802], and (2) the "Bridge" component, a Customer Virtual Local Area Network (C-VLAN) component as defined in IEEE Std 802.1Q-2022 [IEEE22-8021Q].

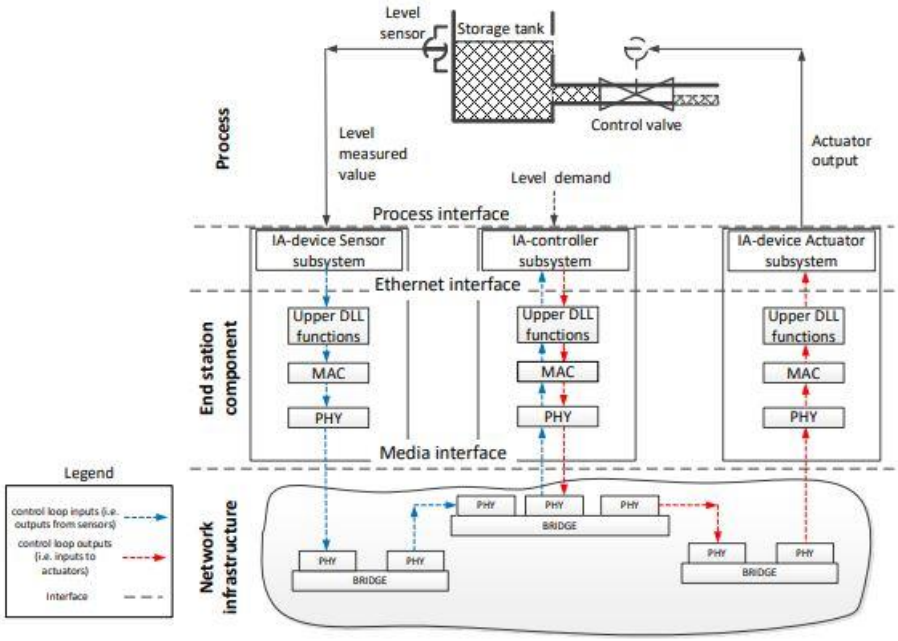

Figure 10: TSN network components as per Figure 1 of IEEE/IEC 60802 D2.1 [IEC/IEEE24-60802]

Early deterministic wireless standardization has intended to build as much as possible on existing functionalities/architectures, therefore the wireless system was modelled like a "virtual node" in the network architecture (i.e., trying to emulate the operation and characteristics of a wired TSN bridge or DetNet router). In such a modelling concept, the radio link of the 6GS becomes an internal entity interconnecting the external ports of the "virtual node", so the radio link is not directly visible from outside. However, due to the fundamental characteristics of the multi-endpoint radio link, the characteristics of such a "virtual node" is heavily affected. The latency of a mobile transmission link is stochastic and heavy-tailed, i.e., larger delay values are more likely compared to exponentially bounded tails, and packet delay variation is relatively large. Therefore, the virtual nodes behave like

having a stochastic backplane, and its properties may lead to extra challenges during designing the end-to-end scheduling.

At the same time, using wireless technology in a Time-Sensitive Network has positive impacts on the latency-related design as well. They are the result of the “wireless by-pass” effect, namely the number of hops between Talker/Listener(s) are significantly reduced due to the wireless link. For example, in usual wired industrial scenarios, the network architecture is based on daisy chaining several bridge components resulting in several 10s of hops network diameter. The proper design of scheduling is needed at each hop (e.g., calculation of GCL at bridges). A limited number of hops means a significant simplification regarding the calculations of schedules in general. In particular, in dynamic systems where schedules need to be adapted, e.g., due to dynamic stream sets (adding/removing streams to/from the schedule), such a simplification is beneficial since it also reduces the time to adapt.

Depending on the use-case scenario, the simplification may end in an extreme scenario, where there is only a single-hop between the endpoints (e.g., an actor is wireless connected, 6G network elements (gNB, UPF) are implemented in the same local Cloud as the industrial controller of the actor.

The following aspects of “wireless by-pass” are investigated:

1. How can transport bottleneck(s) be eliminated?
2. How is inter-stream impact mitigated, such that one stream does not impair the guarantees provided to another stream, e.g., with respect to per-stream delay bounds?

Depending on the actual network scenario the outcome can be extreme, i.e., a simplified TSN architecture with minimal deterministic functions can be defined (e.g., a single TSN-specific service is providing “sync for actors/controllers”, and latency bounds are achieved via simple prioritization and over-dimensioning).

2.6.2 Combined Wireless & Wireline Architecture

The wireline (legacy) architecture has a quite simple partly-meshed topology, that contains ring(s) and daisy-chain of nodes (cf. Figure 11). There are always multiple hops between an “Actor”/”Sensor” and related “application controller(s) (Appl-Ctrl)”. The links between the network nodes have limited bandwidth, therefore scheduling is needed at each hop (including the end station component) (see Section 4 of deliverable D3.1 *6G Convergence Enablers Towards Deterministic Communication Standards* [DET23-D31]).

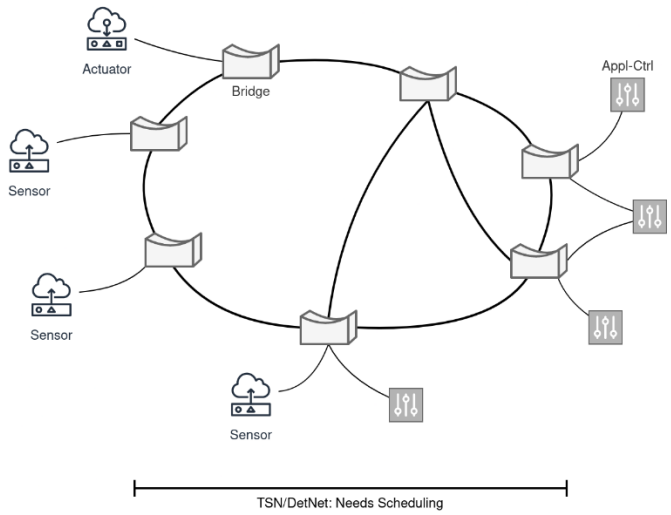

Figure 11: Example of wireline (legacy) architecture

We see two major changes, which are motivated by new scenarios, that impact TSN-based systems and their legacy architecture and lead to a new improved architecture as depicted in Figure 12:

1. Virtualizing and moving applications to Cloud.
2. Adding wireless communication technology.

The first change is moving application controllers to the Cloud, which allows to get rid of specific hardware components and provides flexible compute capacity for them. Using commodity hardware and gaining from Cloud technologies are a big driving force due to possible cost reduction for such changes as well. The internal network of the Cloud domain has a redundant and non-blocking topology. Cloud internal links have high BW capacity, several order of magnitude higher (10 Gbps and beyond) than e.g., legacy industrial networks (100 Mbps).

From latency perspective, within the Cloud it is easy to use over-dimensioning in the TSN/DetNet network design. Careful design of scheduling is needed only in the (legacy) wired part of the end-to-end communication due to the limited BW links.

The second change is about adding wireless access for actors/sensors and improving further a scenario where virtualization and cloudification were already applied. The 6G-RAN/CN components can be implemented close to or within the Cloud. Such a wireless access significantly decreases the number of hops between the actor/sensor and its virtualized controller. However, the wireless hops have a different latency distribution (higher PD and PDV). BW can be easily increased in the system by adding additional radio resources (e.g. densification of radio cells, adding more spectrum), without the need to touch/replace physical interfaces.

The TSN/DetNet-specific design can be simplified, as congestion scenarios can be solved with over-dimensioning both in the radio and Cloud domain. Instead of analyzing the traffic situation/congestion in plenty of hops, the scheduling design can focus on the radio link of the 6G System.

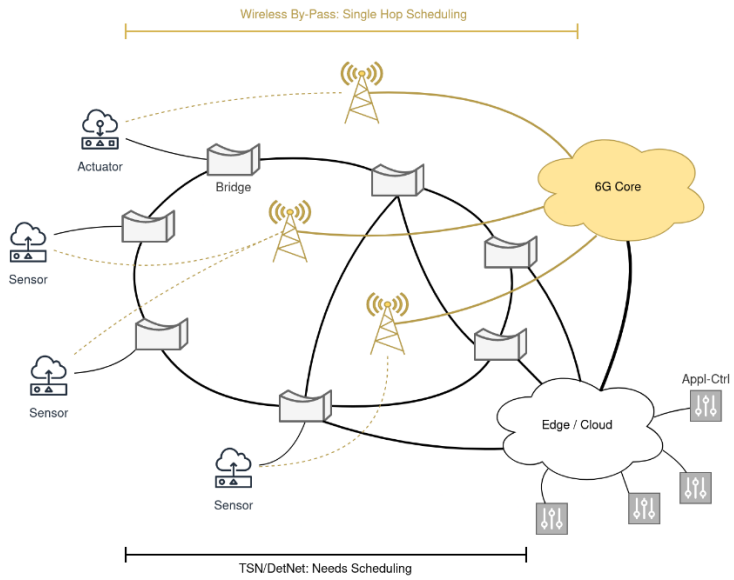

Figure 12: Improved Architecture (adding Wireless and Cloudification)

Depending on the location of the communication partners two scenarios can be distinguished:

1. UE-UE communication with two radio legs. One is UL (uplink) and one is DL (downlink).
2. UE-UPF communication with a single radio leg, either UL or DL.

In case of two radio legs, the radio scheduling must consider both radio transmissions and deal with possible congestions. From the perspective of end-to-end scheduling, the end-to-end (TSN) scheduler sees only a single port-to-port delay distribution to be considered in the calculation of GCLs.

2.6.3 Impact of Improved Architecture

Using wireless links in TSN networks has both negative and positive impacts on the end-to-end system. The PD characteristics of wireless technologies with significantly higher PDV may have a negative impact on end-to-end latency bounds. But there are positive impacts as well, as its usage can significantly decrease the number of network hops (i.e., in extreme cases minimize the connections to a single transport hop). Furthermore, BW update of the whole end-to-end system is simplified and can be controlled by radio design. Of course, wireless and wireline can be combined and finding the best mix of the two technology is use-case dependent.

The most winning scenarios are, where (1) application controllers are moved to the local Cloud; (2) 6G-CORE implemented in the same local Cloud, and (3) actors/sensors are connected using wireless 6G technology. The impact of using an improved architecture needs to be evaluated case-by-case. Algorithms for planning dynamic schedules and adaptation to dynamic stream sets are expected to play a key role to achieve the required end-to-end performance of the transport network.

A detailed evaluation of the impact onto the performance of end-to-end scheduling with respect to schedulability, complexity (runtime) of schedule calculation, etc., will be conducted in future work in WP4.

3 Algorithms for Planning Dynamic Schedules

After we have discussed the system aspects of adapting end-to-end schedules in dynamic systems, which provide the data and mechanisms for adaptation, we now focus on the algorithms for adapting end-to-end schedules in this section. We start with an overview, discuss related work, and then present several approaches for calculating schedules that are (a) robust to (potentially large) PDs and PDVs, and (b) adapt to dynamic stream sets and dynamically changing PD distributions.

3.1 Overview

When talking about adapting end-to-end schedules in dynamic systems, we first need to distinguish between the different causes that lead to dynamic changes that schedules then need to adapt to. We consider the following three different causes, which in turn require different approaches to calculate robust and adaptive end-to-end schedules:

Dynamic stream sets: Streams between talkers and listeners might be added at runtime. Supporting dynamic stream sets is a relatively common goal in research on calculating TSN schedules. So-called incremental scheduling approaches have been proposed in the literature allowing the addition of new streams (and removing old streams), without calculating a new schedule from scratch. Instead, the existing schedule is extended, such that existing streams do not have to be re-scheduled and, therefore, during the transition phase to the new schedule should not be affected (no deadline or latency bound violations during the transition to the new schedule).

Although the problem of incremental scheduling has already received attention in the research community, one of our approaches targeting dynamic packet delay (as discussed in Section 3.4) also lends itself very well to adapt to dynamic stream sets.

Dynamic packet delay distributions: Dynamic characteristics of PD is one of the specific causes for dynamic changes in wireless systems, and therefore, we focus on approaches dealing with this cause. As introduced above, in contrast to the common assumptions in wired TSN systems, we assume PD delay – or more specifically, the port-to-port delay in 6GDetCom nodes – to be of stochastic nature. Therefore, we modelled it in the YANG data model presented in Section 2.3 as histograms, and measure latency at runtime to capture the current PD and predict the future PD using a data-driven approach. Obviously, the quality of the radio channel is affected by network conditions that are not deterministic and known a priori, such as dynamic interference, shadowing by obstacles between mobile station and base station, reflection, diffraction, and scattering on physical objects in the environment, slow and fast fading of the signal, etc. All these factors can lead to dynamic bit error rates, to dynamic frame error rates, and to a dynamic number of retransmissions. Hence, from the perspective of TSN, they all induce stochastic packet delays that may change for different network conditions.

Calculating robust end-to-end schedules for the theoretic worst case with very large PDV would in many situations waste a lot of capacity. For instance, in a GCL, very conservative (long) time intervals would be reserved for streams to cover their worst-case minimum and maximum delay bounds. This directly impacts schedulability and scalability, i.e., the ability to accommodate many streams in the network. Instead, we should consider (a) the actual required reliability instead of aiming for an unrealistic 100 % reliability; (b) the actual PD distribution measured at runtime; (c) changing PD distribution at runtime, ideally with a proactive approach (see Section 2.5) utilizing PD predictions.

We present different approaches to calculate schedules that are robust to PD and can adapt to dynamic PD distributions. One approach tries to maximize the robustness of schedules to PDV. The idea is that such a schedule would avoid adaption as long as possible since it is sufficiently robust to cover a large range of PD – informally speaking, it follows the principle: the best adaptation is the adaptation that you do not have to do. Such an extreme approach might impact schedulability and scalability, as already mentioned since it gravitates towards one extreme end in the solution space (maximum robustness).

Figure 13: Illustration of tardiness in TSN

This motivates the other approaches that we present striving for a given reliability bound typically below 100 %. We present a novel approach mapping the scheduling problem to a graph model with very interesting features, such as the ability to adapt a previously calculated schedule quickly instead of starting all over again from scratch. Moreover, it enables graceful degradation of schedules, where the so-called tardiness (difference between deadline and actual completion time, as shown in Figure 13) is increasing only gradually and bounds are guaranteed with certain, provable reliability (instead of a schedule with arbitrary violations, i.e., no guarantees at all).

In the following, we will describe each of the sketched approaches in detail, after a short discussion of the related work presented next.

3.2 Background and Related Work

We have seen above that there are different goals for adaptivity with respect to end-to-end scheduling, namely, adaptation to dynamic stream sets (incremental scheduling) and dynamic PD distributions. There are many existing approaches for “classic” TSN scheduling in wired networks, which focus on other problems under fundamentally different system assumptions such as dealing with the inherent complexity of standard scheduling problems using fast heuristics, which do not include variable PD, mobility, or similar. For an overview of these approaches, we refer the interested reader to existing surveys like [SOL+23].

Incremental Scheduling

Incremental scheduling, i.e., the ability to add new streams at runtime to the schedule, has received relatively large attention in the research community since this is motivated by popular paradigms like “plug and produce”, where machines can be added dynamically to a shop floor in production. Many approaches focus on the problem of decreasing the runtime of schedule calculation when new streams are added by using heuristics [RPG+17, NDR18, FGD+22]. Possibly the most interesting approaches are those that explicitly target the calculation of extensible schedules that, informally speaking, leave headspace for future streams when calculating the current schedule. In [GRK+21, GRK+23], the authors propose the flexibility curve concept. The basic idea of this approach is to first model the flexibility (number of possibilities to embed streams along paths) of a schedule to add new streams along paths. Based on this model, new streams can be scheduled to minimize the impact on flexibility which allows adding more streams in the future.

Scheduling with Uncertainty in Time-Sensitive Networking

While uncertainty induced by stochastic packet delays remains a blind spot in existing TSN scheduling literature, existing approaches can deal with individual transmission failures or total link failures. For instance, Craciunas et al. [COC+16] show that complete schedule breakdowns may already occur for small timing inaccuracies or sporadic transmission failures. The authors propose *frame isolation constraints* to ensure that such faults do not impair the punctuality guarantees of other streams. While it is straightforward to generalize this approach to include stochastic packet delays, preliminary evaluations show that its induced resource overprovisioning prohibits the scheduler — already for a few wireless streams — from finding eligible solutions.

Similar to our categorization in Section 2.5, existing literature covering total link failures can be grouped into reactive and proactive approaches. For example, a reactive approach is proposed in [PRH18] that computes an alternative route upon detecting a link failure. In contrast, [ZSE+21] develops a proactive approach by modelling switches and transmission links with known failure probabilities.

3.3 Maximize Reliability under Dynamic Packet Delay

In D3.1 “Report on 6G convergence enablers towards deterministic communication standards” [DET23-D31], we already presented a basic wireless-friendly, adaptive end-to-end scheduling algorithm. This scheduling algorithm aims to maximize reliability by maximizing the gap between any two streams. This approach leads to improved results over already existing scheduling algorithms for wired networks. However, this approach only considers the mean value of the estimated packet delays. If this mean delay changes, the reliability of the calculated schedule might suddenly drop and require for a new schedule calculation, which leads to frequent recalculations of the schedule.

As the calculation of new schedules is time-consuming, it is preferable to reduce the need for schedule adaptation in advance. One way to achieve this is by already considering further parameters of the delay measurements and delay predictions as provided by the mechanisms of Section 2.4 and 2.5. To this end, we extend our approach from [DET23-D31] to take into account the known or estimated “uncertainty” of streams.

A detailed description of the variables and constraints of this ILP-based approach can be found in [DET23-D31]. In this report, we extend this approach and focus on the evaluation of different optimization goals. In order to maximize the reliability in our network, we use the $gap_{s,t,e}$ variable as introduced in [DET23-D31] for every combination of two streams s, t and every edge e they have in common on their path. This variable denotes the temporal distance between the completed transmission of one stream and the transmission start of the other stream. The gap variable can be defined by using the binary decision variable $b_{s,t}$ which denotes, whether stream s is scheduled before stream t or vice versa:

$$\begin{aligned} if(b_{s,t}) &\rightarrow gap_{s,t,e} \leq start_{t,e} - end_{s,e} \\ if(\neg b_{s,t}) &\rightarrow gap_{s,t,e} \leq start_{s,e} - end_{t,e} \end{aligned} \quad (3.1)$$

3.3.1 Optimization goals

In the following we present how we can use this gap variable to improve the reliability of calculated schedules. We explain our approaches based on the example network shown in Figure 14.

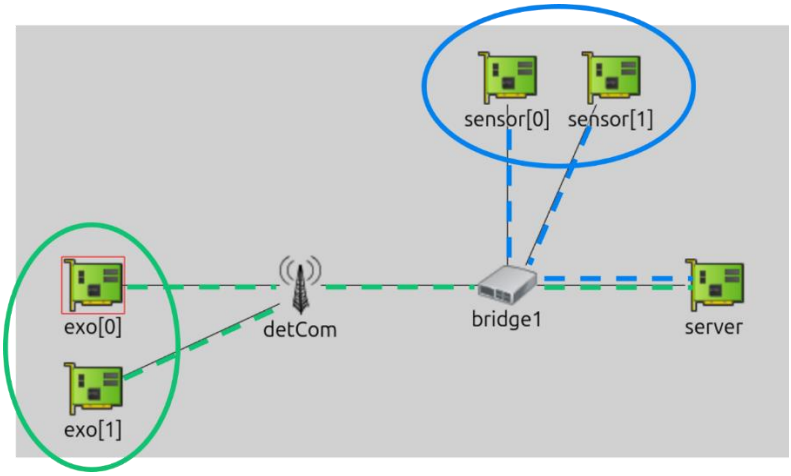

Figure 14: Example network with to wireless and two wired streams

This network consists, for example, of two exoskeletons (exo[0] and exo[1]) which are connected to a server via the 6GDetCom (detCom) node and two sensors (sensor[0] and sensor[1]) with a wired connection to the same server via a TSN bridge (bridge1). There are four streams in this network, $exo_1, exo_2, sensor_1$ and $sensor_2$. All streams transmit data from the respective device to the server connected to bridge1. The exo streams transmit data every $10\ ms$ and need to arrive within their cycle time of $10\ ms$. The $sensor$ streams transmit data every $5\ ms$ and have a maximum end-to-end latency requirement of $100\ \mu s$. Our simulation uses the uplink histogram PD-Wireless-5G-1 from [DET23-D41] with a mean of $\mu = 5.6\ ms$ and a standard deviation of $\sigma = 500\ ns$ for the wireless link within the 6GDetCom node. We assume that all other delays, such as the link delay for wired links and the processing delay of switches are constant.

Optimization Objectives.

As already proposed in D3.1, one approach (further referred to as *Approach* A) to increase the reliability in the network is to maximize the gap between streams by summing up all gap variables $g \in G$ using the following formula:

$$maximize\left(\sum_{g \in G} g\right) \quad (3.2)$$

We have already shown that this approach leads to better results than a scheduling approach solely developed for wired networks. However, this approach has one drawback: In bigger networks, some streams share more edges than other streams. In our example network, this corresponds to the exo streams sharing two edges with each other while the other streams only share one network link. When maximizing the sum of all gaps, this approach increases the time gap between streams with more common edges more than other streams, which is not the desired behavior.

To circumvent this undesired behavior, we propose a new optimization approach (*Approach* B), which maximizes the sum of the smallest gaps between every pair of streams. To this end, we introduce a

new variable $gap_{s,t}$ which denotes the minimal gap between the two streams s and t. We ensure this behavior by introducing a new constraint, which forces this gap variable to be smaller or equal to the gap variable of any edge between s and t:

$$\forall e \in E: gap_{s,t} \leq gap_{s,t,e} \quad (3.3)$$

We denote the set of these per-stream-pair gap variables as G_s. By summing up all gap variables $g \in G_s$, we obtain our new optimization goal, similar as before.

$$maximize\left(\sum_{g \in G_s} g\right) \quad (3.4)$$

Weighting based on uncertainty.

The optimization goal presented above now treats all streams equally and aims to evenly distribute them along the cycle time. Our example network, however, contains two different types of streams with different characteristics. The exo streams suffer from a high PDV as described in our simulation setup above, while the sensor streams don't have any PDV configured, as they are only present in the wired part of the network. The approach presented above, however, does not distinguish between these different types of streams. Thus, it also maximizes the gap between the $sensor$ streams even though they do not influence each other. We aim to make use of the knowledge of different stream characteristics to further optimize our approach.

In the following, we assume our scheduler receives a delay distribution, e.g. by the CNC as described in Section 2.4 and 2.5. We can use this delay distribution to derive a metric of "uncertainty". This uncertainty can hold different values, such as the standard deviation of the delay distribution, the min-max bounds or expected changes to the delay distribution in the future. For the evaluation in this work we utilize the standard deviation of the provided delay distribution.

In our first modification (*Modification* α), we aim to minimize the effect of streams with a higher uncertainty onto streams with lower uncertainty. To this end, we modify our optimization goal to move streams with a higher uncertainty further apart and streams with a lower uncertainty closer together. Our approach first defines a weight parameter $w_{s,z}$ for two streams s,t based on the sum of their standard deviations σ_s and σ_s. Afterwards, the weights are normalized using the Greatest Common Divisor (GCD) of all weights, i.e. we calculate the GCD of all weights and divide all weights by this GCD. This helps to keep the weights small in order to prevent numerical issues in the ILP solver.

In our example network, this leads to the following weights:

$w_{exo1,exo2} = 2;\ w_{sensor1,sensor2} = 0$

$w_{exo1,sensor1} = w_{exo1,sensor2} = w_{exo2,sensor1} = w_{exo2,sensor2} = 1;$

We then use these weights to the objectives from our previous approaches in the following ways:

For *Approach* $A\alpha$ when maximizing the sum of all streams gap variables as in (3.2):

$$maximize\left(\sum_{g_{s,t,e} \in G} w_{s,t} \cdot g_{s,t,e}\right) \quad (3.5)$$

For *Approach* $B\alpha$ when summing up the per-stream-pair gap variables as in (3.4):

$$maximize\left(\sum_{g_{s,t}\in G_s} w_{s,t}\cdot g_{s,t}\right) \quad (3.6)$$

We assume that our α modification leads to better results than the approaches without any modification. However, when two streams suffer from a higher uncertainty, the effect of a collision on those streams might be negligible compared to the uncertainty. To this end, we propose another modification (*Modification* β), which calculates weights based on the similarity of the provided uncertainty, i.e. when two streams have a similar uncertainty, the weight is low and when the uncertainty differs, the weight is high. For this approach, we calculate the weight $w_{s,t}$ by calculating the absolute difference of the standard deviations $|\sigma_s - \sigma_t|$ and again normalizing it with the GCD of all weights. This leads to the following weights for our example network:

$w_{exo1,exo2} = w_{sensor1,sensor2} = 0$

$w_{exo1,sensor1} = w_{exo1,sensor2} = w_{exo2,sensor1} = w_{exo2,sensor2} = 1;$

3.3.2 Evaluation

For our evaluation of the approaches presented above, we use networks structured as our example network from Figure 14 but with a varying number of network devices up to 150 total devices. The packet size for each stream is randomly chosen to be between $16\ B$ and $512\ B$. We use a scheduling algorithm for a wired network as our baseline approach ($Approach\ 0$).

Figure 15 shows the percentage of late packets between our different approaches. We can see that all our wireless-friendly optimization goals are able to improve upon the baseline scheduling approach for wired networks. $Approach\ A$ mainly reduces the percentage of late packets for the worst streams while $Approach\ B$ is also able to improve the mean and median percentage of late streams. We can also see that considering further information about the expected uncertainty of a stream with $Modifications\ \alpha$ and β improves the percentage of late packets even further.

In Figure 16 we can see that our $Approach\ B$ is also able to reduce the arrival time jitter for the sensor streams.

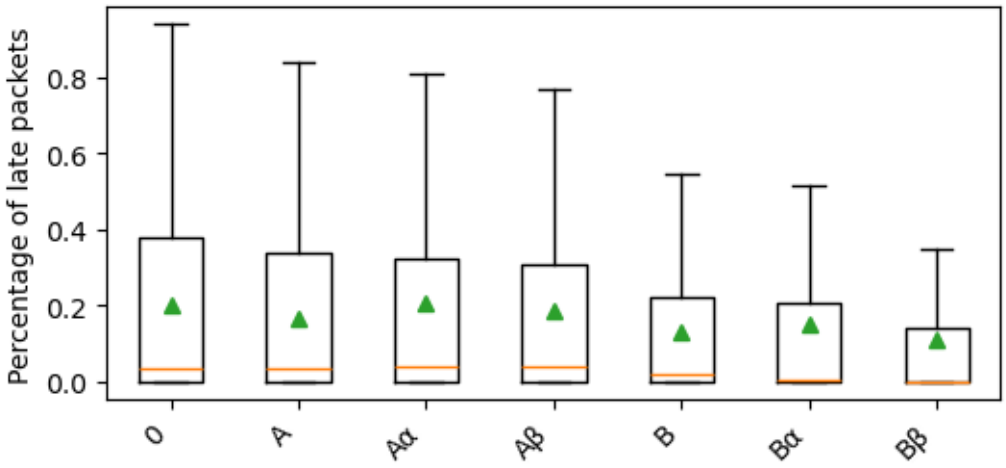

Figure 15: Percentage of late packets of sensor streams per approach

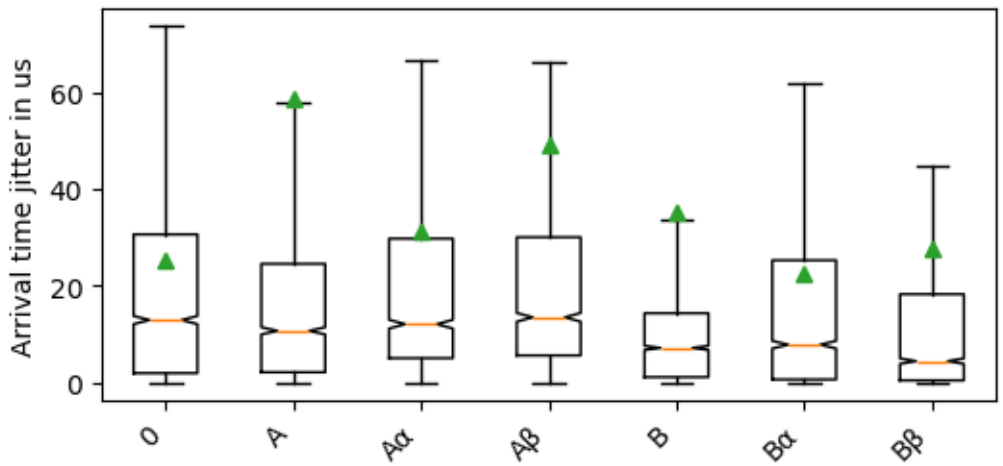

Figure 16: Average arrival time jitter of sensor streams per approach

Our evaluation shows that choosing different optimization goals clearly improves upon scheduling approaches created for wired networks. Additionally, it suggests that utilizing further “uncertainty” metrics reduce the number of late packets even further which can be used as an advantage to defer the need for adapting schedules.

3.4 Adaptation to Dynamic Packet Delays

While maximizing the reliability against dynamic packet delays, as discussed in Section 3.3, can defer the need to adapt the schedule to some degree, employing this approach in a stand-alone manner has two disadvantages:

1. Reliability guarantees can only be made after the schedule has been computed. That is, the approach of Section 3.3 aims to maximize the achieved reliability guarantees while the streams’ end-to-end latency requirements are satisfied. Instead, the strategy described in this section conserves the reliability guarantees of each stream and aims to optimize the schedule’s maximum tardiness.

2. Section 3.3 cannot circumvent the need for adaptation entirely. For example, Figure 17 shows the wireless transmission interval for a single frame f across a wireless link $[u, v]$. While the corresponding schedule may achieve a reliability guarantee of 33.33% for f, its reliability already plummets to below 1.5% when the uplink histogram is only shifted by one millisecond to the right.

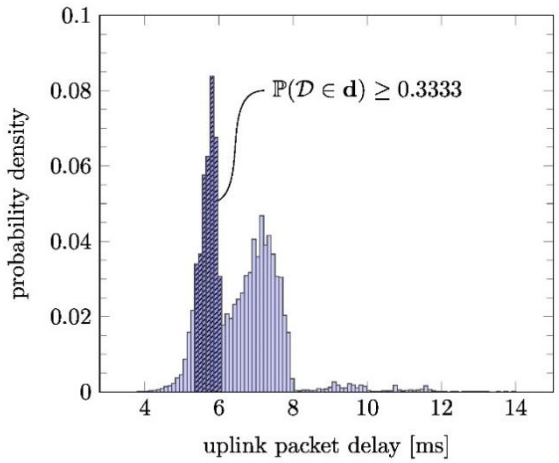

Figure 17a: Before degradation

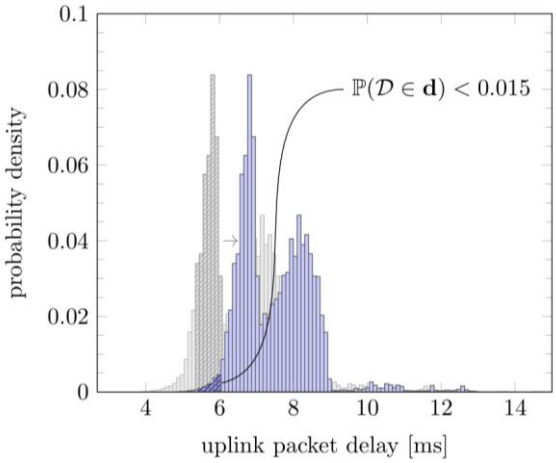

Figure 17b: After degradation

Figure 17: Reliability impairment caused by wireless channel degradation

To address both shortcomings, we devise a linear-time adaptation strategy to eliminate reliability impairments of Figure 17b. Section 3.4.1 introduces shuffle graphs as a natural graphical representation of schedules in wireless TSN. Thereafter, Section 3.4.2 introduces a simple adaptation strategy that, based on shuffle graphs, can adapt the schedule in linear time. Finally, Section 3.4.3 and Section 3.4.4 illustrates the impact of our proposed adaptation strategy and discusses its implications on the Deterministic6G infrastructure. For additional details, we refer the reader to [Egg24].

Remark: To improve visualization, we use rather small reliability requirements of, for example, 33.33% throughout this section. The presented concepts can, however, be applied analogously for requirements that far exceed 99%.

3.4.1 Shuffle Graphs as a Graphical Representation of TSN Schedules

We devise a graphical representation of TSN schedules to enable a linear-time adaptation strategy. To this end, we adjust the Disjunctive Graph Model (DGM) — a well-established tool from job-shop scheduling — to incorporate exactly the scheduling capabilities of TSN bridges.

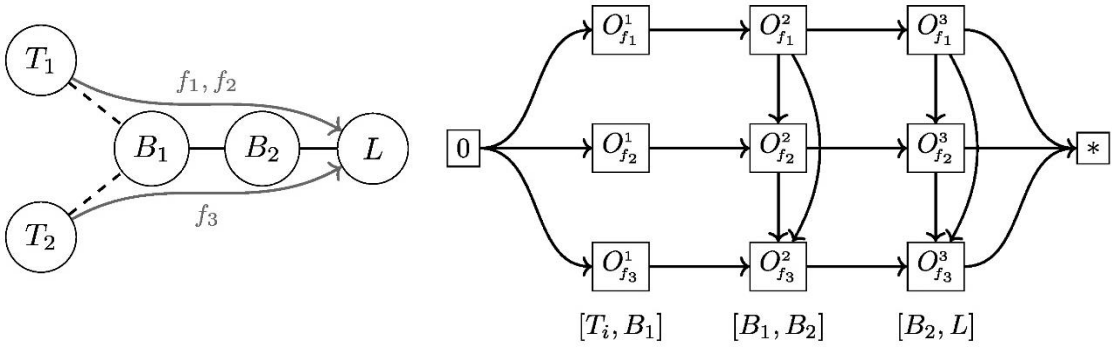

Figure 18a: Network topology Figure 18b: Transmission Order

Figure 18: Simple network (a) and transmission order determined by a scheduler (b)

Disjunctive Graph Selections. In the first step, the scheduler is employed to compute the Gate Control Lists (GCL) for each egress port of each TSN bridge. While schedulers typically discard additional information that is not encoded in the GCLs, we require the scheduler to further define the intended transmission order of each egress port. For example, Figure 18a shows a simple network topology with three message streams $\{f_1, f_2, f_3\}$ that traverse the network from mobile talkers T_1, T_2 to the listener L. By computing an IEEE 802.1Qbv schedule to configure the GCLs, the scheduler implicitly defines the transmission order, i.e., the disjunctive graph selection, as in Figure 18b. This selection encodes two vital types of constraints:

- $O^1_{f_1} \rightarrow O^2_{f_1}$ encodes that the second transmission of f_1(i.e., via $[B_1, B_2]$) can only commence after the first transmission of f_1 (i.e., via $[T_1, B_1]$) completed.
- $O^2_{f_1} \rightarrow O^2_{f_2}$specifies the order of the contesting transmissions of $f_1$and f_2 via $[B_1, B_2]$.

Discussion. While [Egg24] contains the formal definition of disjunctive graphs and consistent selections, this document addresses the central considerations from a system design perspective:

How do disjunctive graphs cover the scheduling capabilities of TSN bridges? Reviewing Figure 18b, we illustrate that the IEEE 802.1Qbv capabilities do not support arbitrary transmission orders. For instance, if both $f_1$and f_2 share the same Priority Code Point (PCP), bridge B_1must transmit $f_1$first if and only if f_1 is enqueued first. Consequently, we constrain that $O^2_{f_1} \rightarrow O^2_{f_2}$ and $O^3_{f_1} \rightarrow O^3_{f_2}$ share the same orientation. Furthermore, disjunctive graphs must not induce any cyclic dependencies between transmissions.

How do disjunctive graphs support modelling OFDMA? Compared to mutual exclusion constraints that are typically employed for scheduling in wired TSN, disjunctive graphs also support modelling OFDMA by virtually duplicating wireless links. For example, T_1 may employ different frequency bands for transmitting $f_1$and f_2, eliminating the streams' temporal transmission contest. By duplicating the wireless link $[T_1, B_1]$ for each available frequency band, the scheduler must specify the allocated Resource Blocks (RB) for each wireless transmission. Consequently, if the frequency bands for $f_1$and

f_2 are non-overlapping, the disjunctive graph in Figure 18b does not include an edge $O^1_{f_1} \rightarrow O^1_{f_2}$, implying that $f_1$and f_2 can be transmitted concurrently over $[T_1, B_1]$.

How do disjunctive graphs compare to alternative modelling approaches? Disjunctive graphs provide a higher degree of modelling freedom over existing approaches. In particular, typical approaches often consider frame-based transmission orderings, where a frame f_1 is transmitted over [B, B'] before f_2 if and only if $f_1$is transmitted over all joint links before f_2. Such coarse-grained approaches appear unfavorable in more complex network topologies that contain diamond structures (such as in Figure 19). In contrast, disjunctive graphs provide full decision-flexibility to the scheduler.

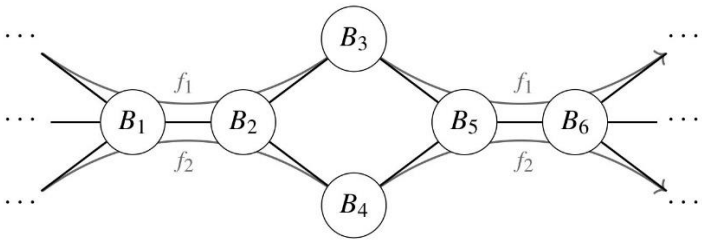

Figure 19: Diamond network

Shuffle Graphs. While disjunctive graph selections encode the transmission order for each egress port, we derive shuffle graphs as a one-to-one graphical representation of a wireless TSN schedule. For instance, Figure 20 shows the corresponding shuffle graph of Figure 18b. In this simplified setting, the shuffle graph is derived in two steps:

1. *Add FIFO edges* to isolate potential frame transmission faults. That is, for each edge of the form $O^2_{f_1} \rightarrow O^2_{f_2}$ — specifying that B_1 transmits f_1 before f_2 via $[B_1, B_2]$ — the shuffle graph adds the FIFO edge $O^2_{f_1} \rightarrow O^1_{f_2}$ to ensure that f_2 does not arrive too early at B_1. Otherwise, it may occur that f_2's early arrival at B_1 causes f_2 to "steal" the transmission slot of f_1, potentially impairing the reliability of f_2 itself (when dropped by PSFP at B_2) or the reliability of other high-criticality streams due to interference at subsequent egress queues.
2. *Add transmission weights* to separate the transmission offsets accordingly: $w\left(0 \rightarrow O^j_{f_i}\right)$ corresponds to the release time of f_i at its talker. $w\left(O^j_{f_i} \rightarrow O^{j+1}_{f_i}\right)$ specifies an upper bound for f_i's transmission over its jth hop, consisting of processing, propagation, and transmission delay. For example, $w\left(O^1_{f_1} \rightarrow O^2_{f_1}\right) = 5.966ms$ equals the upper bound of Figure 17a to achieve a reliability of 33.33%, whereas $w\left(O^2_{f_1} \rightarrow O^3_{f_1}\right) = 1.5ms$ conservatively bounds the delay of f_1 via wired Ethernet links.[5] Finally, we set $w\left(O^j_{f_i} \rightarrow O^{l-1}_{f_k}\right) = d^{max}\left(O^j_{f_i}\right) - d^{min}(O^{l-1}_{f_k})$ to isolate transmission faults. For example $w\left(O^2_{f_1} \rightarrow O^1_{f_2}\right) = 1.5ms - 5.348ms = -3.848ms$ ensures that f_2 only arrives at B_1 after the transmission of f_1 via $[B_1, B_2]$ completed.

[5] We choose $1.5ms$ to increase visibility in Figure 22. A more realistic setting may set $w\left(O^2_{f_1} \rightarrow O^3_{f_1}\right) = 12\ \mu s$ to include the transmission delay $\frac{100byte}{100\ Mbit\ s^{-1}} = 8\mu s$ and to bound the processing and propagation delay by $4\mu s$.

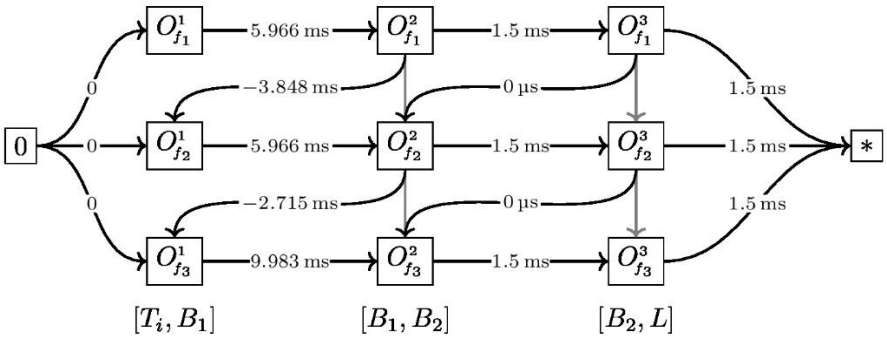

Figure 20: Shuffle graph induced by the selection of Figure 18b

Deriving TSN Configurations. Shuffle graphs directly encode the GCL and PSFP configuration for each TSN bridge and each 6GDetCom node, as shown in Figure 20. In particular, the longest path $C(O^j_{f_i})$ from source 0 to an operation $O^j_{f_i}$ corresponds to the transmission offset of f_i's transmission over its jth hop. The GCL of $[B_1, B_2]$ then contains an entry $[o, c]$ for f_2's transmission, where the gate

- opens at time $o = S^{min}([B_1, B_2], f_2) = C(O^2_{f_2}) = 2 \times 5.966ms - 3.848ms = 8.062ms$, and
- closes at time $c = S^{max}([B_1, B_2], f_2) = C(O^2_{f_2}) + 1.5ms = 9.562ms$.

Analogously, we define PSFP enforced intervals that discard any frames that arrive outside their specification. In case of f_2's transmission via $[T_1, B_1]^2$, the transmission may last between $5.348ms$ and $5.966ms$ to achieve a reliability of 33.33% (see Figure 17a). We therefore configure the PSFP enforced interval to forward f_2 at B_1 if and only if it arrives within $[R^{min}(B_1, f_2), R^{\max}(B_1, f_2)]$, where

- $R^{min}(B_1, f_2) = (S^{min} + d^{min})([T_1, B_1], f_2) = C(O^1_{f_2}) + 5.348ms = 7.466ms$, and
- $R^{max}(B_1, f_2) = R^{min}(B_1, f_2) + 5.966ms - 5.348ms = 8.084ms$.

In case f_2 does not arrive at B_1 within said interval, B_1 must either discard f_2 or demote f_2 (e.g., to best-effort).

To realize a traversal of the shuffle graph that is linear in the number of transmission operations, it is important to avoid traversing implicit edges. For instance, Figure 20 no longer contains any edge from f_1's operations to f_3's operations. This is because a any FIFO edge like $O^2_{f_1} \to O^1_{f_3}$ can be substituted by the existing path $O^2_{f_1} \to O^1_{f_2} \to O^2_{f_2} \to O^1_{f_3}$.

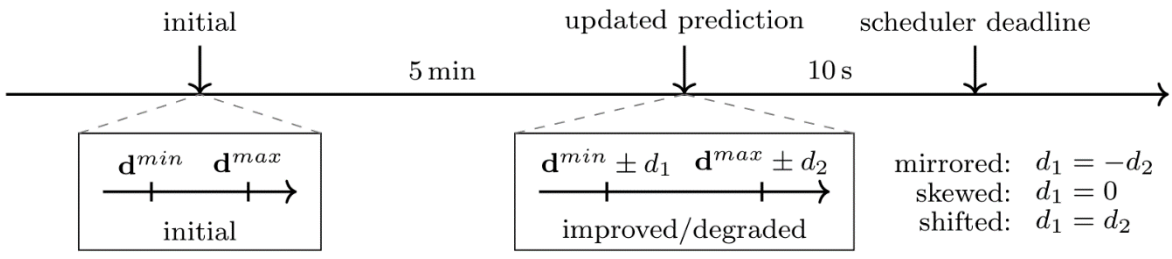

Figure 21: Simulated timeline for adapting the end-to-end TSN schedule

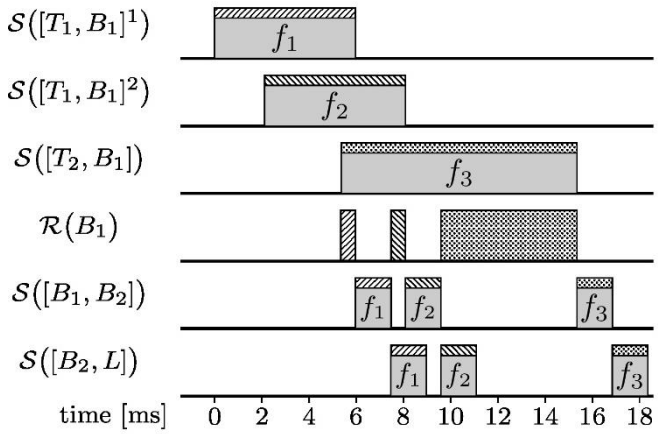

Figure 22: Schedule encoded by shuffle graph of Figure 20

3.4.2 Linear-Time Adaptation Strategy

Shuffle graphs enable a simple adaptation strategy to uphold the reliability guarantees even for degrading wireless channels as in Figure 17b. We consider a timeline as depicted in Figure 21: For an initial PD distribution, the scheduler is allowed to find and optimize a TSN schedule for a relatively long time (here, for five minutes). After that, the prediction service notifies the end-to-end scheduler of updated PD distributions which invalidates the initially computed TSN schedule. The scheduler then only has a short timespan (here, ten seconds) to find an adapted solution before the new TSN configuration is due. For the adapted transmission intervals, we consider different degradation patterns (mirrored, skewed, shifted), which are used in Section 3.4.4 to evaluate our adaptation strategy.

Shuffle graphs allow for a highly efficient adaptation strategy with a runtime that is linear in the number of frame transmissions per hypercycle. This is achieved by first adjusting the reserved transmission intervals for the updated PD distributions, e.g., from $[5.348ms, 5.966ms]$ to $[6.348ms, 6.966ms]$ to regain a reliability guarantee of 33.33% in Figure 17. Subsequently, a single linear-time traversal of the shuffle graph suffices to update its weighted edges and to derive the adapted TSN configuration, as described in Section 3.4.1.

Reviewing Figure 22, we can analyze the impact of adapting the schedule to accommodate for the degraded wireless channel. Assuming for simplicity that only the transmission of f_1 is affected, the adapted schedule would need to shift all remaining transmissions by $1ms$ forward in time to retain a robust schedule. Consequently, our adaptation strategy converts streams' reliability impairments (as in Figure 17b) into tardiness where the schedule may no longer satisfy all end-to-end latency requirements of the message streams. By symmetry, however, we find that our adaptation strategy converts streams' reliability surpluses (of improving wireless channels) into additional slack that provides lower-priority traffic with additional transmission opportunities. In latter case, our adaptation strategy thus allows for an almost immediate redeployment of the new TSN configuration

3.4.3 Evaluation

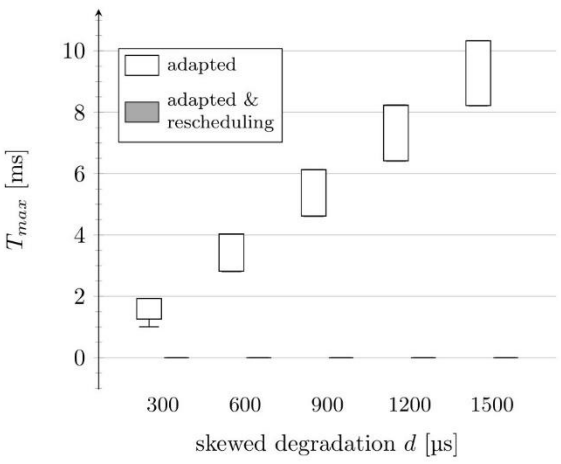

Figure 23a: 20 wireless streams

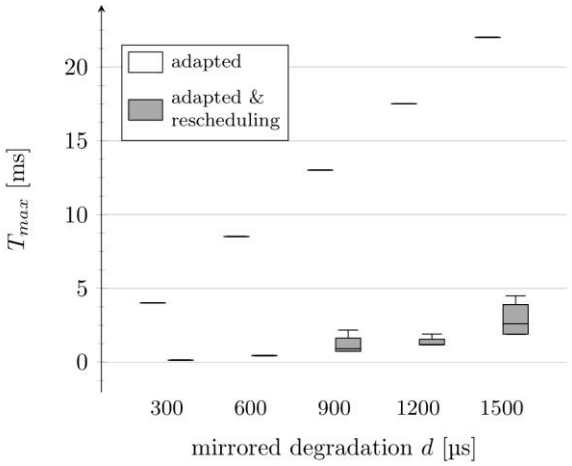

Figure 23b: 60 wireless streams

Figure 23: Maximum tardiness (T_{max}) after adapting the schedule as in Section 3.4.2 (white box plots) and after additional rescheduling for 10 seconds (grey box plots)

As discussed in Section 3.4.2, our adaptation strategy converts reliability impairments of degrading wireless channels into tardiness of the adapted schedule. To evaluate the impact of this conversion, we consider three degradation patterns that modify the initial transmission interval $[d^{min}, d^{max}]$ via

- shifting $[d^{min} + d, d^{max} + d]$,
- skewing $[d^{min} + 0, d^{max} + d]$, and
- mirroring $[d^{min} - d, d^{max} + d]$.

For example, Figure 17b corresponds to a shifted degradation with $d = 1ms$.

We consider a simple network that consists of two wired partitions. Each partition comprises 15 TSN bridges in a tree topology where the root bridges are connected by a wireless link. We consider fixed wireline traffic of 50 frames per hypercycle that stay within their respective partition but require a short end-to-end latency of $200\mu s$ and zero arrival jitter at their listeners. In comparison, we vary the number of wireless streams that utilize the wireless link to traverse across both partitions. Each wireless stream requires a reliability of 33.33% to arrive at their listeners within an end-to-end latency of $10ms$ and a jitter of $5ms$ [DET23-D11].

Figure 23 shows the results for skewed degradation patterns with 20 wireless streams (a) and for mirrored degradation patterns with 60 wireless streams (b). We evaluate the maximum tardiness T_{max} of the adapted schedule, i.e., the maximum delay between actual and expected arrival experienced by wireless or wireline streams. Figure 23 shows an expected linear increase in the maximum tardiness for degrading wireless channels.

By reapplying the metaheuristic scheduler of [Egg24] for a short period of time ($\leq 10s$), the adapted schedule can be improved drastically. For Figure 23a, rescheduling is even able to eliminate the induced tardiness entirely, allowing to continuously guarantee the streams' end-to-end latency requirements. Figure 23b illustrates, however, that while rescheduling is highly effective, it cannot

provide any formal guarantees about finding an optimal scheduling within an arbitrarily small period. In the following, we therefore discuss the implications of our adaptation strategy to the Deterministic6G infrastructure.

3.4.4 Discussion

Our summarized findings show that shuffle graphs provide a linear-time adaptation strategy to

i. convert reliability impairments of degrading channels into tardiness, and to
ii. convert reliability surpluses of improving channels into additional slack.

To further reduce the tardiness of (i), we find that the metaheuristic rescheduling of [Egg24] is highly effective but cannot provide any formal guarantees about finding an optimal solution within a small period of time. We therefore argue that the reliability contracts between upper-layer services and the infrastructure provider should specify a graceful degradation of end-to-end latency guarantees for a wide range of channel degradation patterns. For instance, these contracts should specify a worst-case degradation scenario beyond which no formal end-to-end guarantees can be made. Still (i) and (ii) can be utilized to define optimal or near-optimal graceful degradation bounds in the reliability contracts.

While (i) and (ii) may be employed in a reactive scheduling approach where the channel prediction alerts the CNC of improving or degrading channel qualities, it is also possible to employ (ii) for proactive scheduling mechanisms. More specifically, a proactive approach initially computes a sequence of eligible schedules $(S_i)_{i=1}^{n}$for accompanying channel predictions $(P_i)_{i=1}^{n}$. Then, for an updated channel prediction P_{n+1} with improved wireless channel states over P_j $(1 \leq j \leq n)$, (ii) can be utilized to efficiently transform S_j into an eligible schedule S_{n+1} that provides additional transmission opportunities to lower-priority traffic.

3.5 Adaptation to Dynamic Stream Sets

Section 3.4 adapted the schedule to degrading or improving wireless channels for a fixed stream set. We now extend the adaptation strategy to incorporate joining and leaving message streams. Similar to Section 3.4, the extended adaptation strategy consists of two phases:

1. Employ an initial heuristic H to adapt the schedule.
2. Reapply the metaheuristic scheduler of [Egg24] to reduce the schedule's tardiness.

This section focuses on devising suitable initial heuristics to account for dynamic stream sets. To this end, we identify two central requirements: First, H should be fast, i.e., instead of relying on an exhaustive search to find an optimal solution, it should leave optimizing the schedule to the second phase. Second, H should avoid modifying the schedule for streams that are unaffected by the dynamic stream set change to ease configuring the TSN bridges with the updated gate control lists.

More concretely, the initial heuristic H receives the initial stream set F which is subsequently modified by removing the leaving streams F_l and by adding the joining streams F_j, i.e., $F \leftarrow (F \setminus F_l) \cup F_j$. We start by removing F_l (see Section 3.5.1) to introduce additional gaps in the schedule that can be filled by adding F_j (see Section 3.5.2).

3.5.1 Adaptation to Leaving Streams

Employing the shuffle graphs of Section 3.4.1 to represent wireless TSN schedules, we devise a simple adaptation strategy that replaces transmission operations $O_{f_i}^j$ of leaving frames $f_i \in F_l$ with placeholders $O_{p_i}^j$ (see Figure 24). When deriving the TSN configuration thereafter, the placeholders are used to retain the original transmissions of $F \setminus F_l = \{f_1, f_3\}$ while no GCL entries are set for $O_{p_2}^j$.

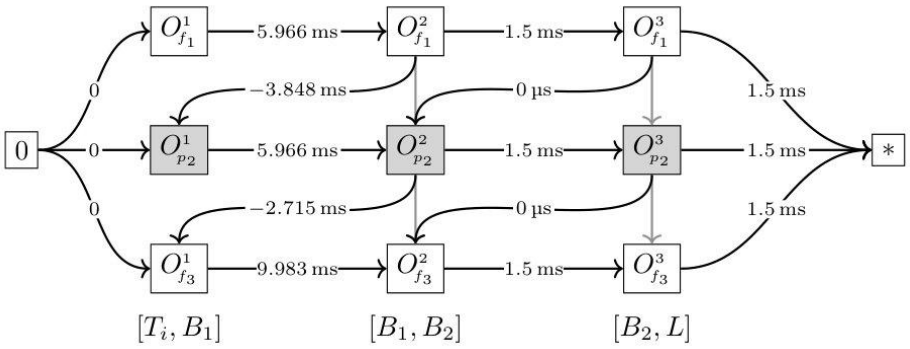

Figure 24: Shuffle graph after stream f_2 is removed and replaced by a placeholder p_2

From a practical perspective, it is not required to directly compute a TSN configuration for the intermediate step $F \setminus F_l$. It is therefore sufficient to simply mark each $f \in F_l$ as removed, indicating the subsequent routines to handle each operation O_f^j as a placeholder. This process can therefore be completed within time $\#F_l$. Still, to avoid traversing an overwhelming amount of placeholder operations in the shuffle graph, it is advisable to merge neighboring placeholders that share the same transmission link.

3.5.2 Adaptation to Joining Streams

For every joined stream $f \in F_j$, we sequentially determine the schedule of each hop in f's route. That is, for each (unscheduled) operation O_f^i, we initially compute the current transmission order $f_1 \rightarrow f_2 \rightarrow \cdots \rightarrow f_k$ of already scheduled frames. Thereafter, the heuristic H aims to find a suitable position j where f should be inserted, i.e., yielding the updated transmission order $f_1 \rightarrow \cdots f \rightarrow f_j \rightarrow \cdots f_k$ at the ith transmission hop of f's route.

We can define a simple upper bound for the position j as follows: f must not be scheduled after f_k if f's resulting transmission time is after its effective deadline. The effective deadline of O_f^i is determined by subtracting f's remaining transmission delays from its deadline at the listener. In other words, if f would be transmitted at its ith hop after f_k, it would no longer be able to satisfy its end-to-end latency requirement.

To determine a lower bound for position j, we include the requirement that H should avoid modifying the schedule for unaffected streams $f_k \in F \setminus F_l$. Hence, f may only be inserted before f_k if the transmission offset of f_k is not delayed, e.g., there exists a sufficiently large placeholder between f_{k-1} and f_k in the shuffle graph.

4 Conclusions & Future Work

In this report, we have presented system concepts and algorithms to implement dependable end-to-end communication in dynamic systems of wired and, in particular, wireless network elements (wireless TSN bridges called 6GDetCom nodes). With respect to system concepts, we presented:

- A YANG data model to model dynamic stochastic packet delay (port-to-port delay). This model provides the information required to calculate and adapt end-to-end schedules to the network control logic (algorithms) executed by the CNC.
- Proactive and reactive approaches based on the NETCONF protocol to trigger the adaptation of end-to-end schedules and reduce the time while which the end-to-end schedule might be broken during the transition phase.
- The wireless by-pass characteristic to enhance path redundancy and reduce the complexity of end-to-end scheduling.
- Different algorithms to (a) maximize reliability under dynamic packet delay, (b) enable fast adaptation by modifying existing schedules (instead of starting from scratch), (c) enable graceful degradation under degrading packet delay.

Altogether, these novel concepts make a large step towards dependable communication in dynamic wireless 6G systems that goes significantly beyond the assumptions made for “deterministic” wired systems.

As a next step, we relax the assumption to operate in a single domain with a global view onto all system parameters and consider multi-domain systems. In such multi-domain systems, it will become crucial to divide the control among the different domains of a larger or administratively distributed system. Obviously, this step from a single fully centralized system to multiple interacting systems poses new challenges for end-to-end scheduling spanning now paths over multiple domains. This leads to new questions such as how to split the end-to-end budget of end-to-end delay between the different domains and how to “divide-and conquer” the end-to-end system?

References

[3GPP23-22261]	3GPP TS 22.261, Service Requirements for the 5G System, v19.4.0, 2023
[3GPP24-23501]	3GPP TS 23.501, System architecture for the 5G System (5GS), v18.5.0, 2024
[3GPP24-28552]	3GPP TS 28.552, Technical Specification Group Services and System Aspects, Management and orchestration; 5G performance measurements, V18.6.0, 2024
[COC+16]	S. S. Craciunas, R. S. Oliver, M. Chmelík, W. Steiner: Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks. In: Proceedings of the 24th International Conference on Real-Time Networks and Systems (RTNS 2016), 2016, DOI: 10.1145/2997465.2997470
[Del04]	T. DelSole: Predictability and Information Theory. Part I: Measures of Predictability, Journal of the Atmospheric Sciences, 61(20), October 2004, DOI: 10.1175/1520-0469(2004)061<2425:PAITPI>2.0.CO;2
[DET23-D11]	DETERMINISTIC6G, Deliverable 1.1, DETERMINISTIC6G Use Cases and Architecture Principles, June 2023

[DET23-D21]	DETERMINISTIC6G, Deliverable 2.1, First report on 6G centric enabler, Dec. 2023
[DET23-D31]	DETERMINISTIC6G, Deliverable 3.1, Report on 6G convergence enablers towards deterministic communication standards, Dec. 2023
[DET23-D41]	DETERMINISTIC6G, Deliverable 4.1, DETERMINISTIC6G DetCom simulator framework release 1, Dec. 2023
[DET24-D12]	DETERMINISTIC6G, Deliverable 1.2, First report on DETERMINISTIC6G architecture, April 2024
[DET24-D42]	DETERMINISTIC6G, Deliverable 4.1, Latency measurement framework, March 2024
[Egg24]	Simon Egger, Adaptive Robustness in Wireless Time-Sensitive Networks (TSN), Master's Thesis, University of Stuttgart, April 2024
[FGD+22]	J. Falk , H. Geppert , F. Dürr, S. Bhowmik, K. Rothermel: Dynamic QoS-Aware Traffic Planning for Time-Triggered Flows in the Real-Time Data Plane, IEEE Transactions on Network and Service Management, 19(2), June 2022, DOI: 10.1109/TNSM.2022.3150664
[GRK+21]	C. Gärtner, A. Rizk, B. Koldehofe, R. Hark, R. Guillaume. R. Steinmetz: Leveraging Flexibility of Time-Sensitive Networks for dynamic Reconfigurability. In: Proceedings of the 2021 IFIP Networking Conference (IFIP Networking), June 2021, DOI: 10.23919/IFIPNetworking52078.2021.9472834
[GRK+23]	Christoph Gärtner, A. Rizk, B. Koldehofe, R. Guillaume, R. Kundel, R. Steinmetz: Fast Incremental Reconfiguration of Dynamic Time-Sensitive Networks at Runtime. Computer Networks: The International Journal of Computer and Telecommunications Networking, 224(C), Apr 2023, DOI: 10.1016/j.comnet.2023.109606
[IEEE14-802]	802-2014, IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture, 2014, DOI: 10.1109/IEEESTD.2014.6847097
[IEEE15-8021Qbv]	802.1Qbv-2015, IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic, 2015, DOI: 10.1109/IEEESTD.2016.8613095
[IEEE17-8021Qci]	802.1Qci-2017, IEEE Standard for Local and metropolitan area networks--Bridges and Bridged Networks – Amendment 28: Per-Stream Filtering and Policing, 2017, DOI: 10.1109/IEEESTD.2017.8064221
[IEEE18-8021Qcc]	802.1Qcc-2018, IEEE Standard for Local and Metropolitan Area Networks--Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements, 2018, DOI: 10.1109/IEEESTD.2018.8514112
[IEEE22-8021Q]	802.1Q-2022, IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks, 2022, DOI: 10.1109/IEEESTD.2022.10004498
[IEC/IEEE24-60802]	IEC/IEEE 60802, Time-Sensitive Networking Profile for Industrial Automation, Draft 2.2, April 2024
[IETF06-RFC4253]	Internet Engineering Taskforce (IETF), RFC 4252, The Secure Shell (SSH) Authentication Protocol, 2006
[IETF11-RFC6101]	Internet Engineering Taskforce (IETF), RFC 6101, The Secure Sockets Layer (SSL) Protocol Version 3.0, 2011

[IETF11-RFC6241]	Internet Engineering Taskforce (IETF), RFC 6241, Network Configuration Protocol (NETCONF), June 2011
[IETF16-RFC7950]	Internet Engineering Taskforce (IETF), RFC 7950, The YANG 1.1 Data Modeling Language, August 2016
[IETF18-RFC8446]	Internet Engineering Taskforce (IETF), RFC 8446, The Transport Layer Security (TLS) Protocol Version 1.3, August 2018
[IETF19-RFC8639]	Internet Engineering Taskforce (IETF), RFC 8639, Subscription to YANG Notifications, September 2019
[IETF19-RFC8641]	Internet Engineering Taskforce (IETF), RFC 8641, Subscription to YANG Notifications for Datastore Updates, September 2019
[IETF99-RFC2246]	Internet Engineering Taskforce (IETF), RFC 2246, The TLS Protocol, Version 1.0, 1999
[MNS+23]	S. Mostafavi, R. Nandan, G. P. Sharma, J. Gross: Latency Probability Prediction for Wireless Networks: Focusing on Tail Probabilities. In Proceedings of IEEE Global Communications Conference (GLOBECOM 2023), December 2023
[MSG23]	S. Mostafavi, G. P. Sharma, J. Gross: Data-Driven Latency Probability Prediction for Wireless Networks: Focusing on Tail Probabilities, arXiv:2307.10648, 2023, DOI: 10.48550/arXiv.2307.10648
[MTS+24]	S. Mostafavi, M. Tillner, G. P. Sharma, J. Gross: EDAF: An End-to-End Delay Analytics Framework for 5G-and-Beyond Networks. In Proceedings of the 11th International Workshop on Computer and Networking Experimental Research using Testbeds (CNERT 2024) at INFOCOM, May 2024, DOI (arXiv pre-preprint): 10.48550/arXiv.2401.09856
[NDR18]	N. G. Nayak, F. Dürr, and K. Rothermel: Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks, IEEE Transactions on Industrial Informatics, 14(5), May 2018, DOI: 10.1109/TII.2017.2782235
[PRH18]	F. Pozo, G. Rodriguez-Navas, H. Hansson: Schedule Reparability: Enhancing Time-Triggered Network Recovery Upon Link Failures. In: 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2018, DOI: 10.1109/RTCSA.2018.00026
[RPG+17]	M. L. Raagaard, P. Pop, M. Gutiérrez, W. Steiner: Runtime Reconfiguration of Time-Sensitive Networking (TSN) Schedules for Fog Computing. In: Proceedings of the 2017 IEEE Fog World Congress (FWC), Santa Clara, CA, 2017, DOI: 10.1109/FWC.2017.8368523
[SOL+23]	T. Stüber , L. Osswald , S. Lindner, M. Menth: A Survey of Scheduling Algorithms for the Time-Aware Shaper in Time-Sensitive Networking (TSN), IEEE Access, 11, 2023, DOI: 10.1109/ACCESS.2023.3286370
[SPS+23]	G. P. Sharma, D. Patel, J. Sachs, M. De Andrade, J. Farkas, J. Harmatos, B. Varga, H. -P., Bernhard, R. Muzaffar, M. Ahmed, F. Dürr, D. Bruckner, E.M. De Oca, D. Houatra, H. Zhang and J. Gross: Toward Deterministic Communications in 6G Networks: State of the Art, Open Challenges and the Way Forward, IEEE Access, vol. 11, pp. 106898-106923, 2023, DOI: 10.1109/ACCESS.2023.3316605
[W3C99-XPath]	World Wide Web Consortium (W3C): XML Path Language (XPath), Version 1.0, 1999

[ZSE+21]	Y. Zhou, S. Samii, P. Eles, Z. Peng: Reliability-Aware Scheduling and Routing for Messages in Time-Sensitive Networking. ACM Trans. Embed. Comput. Syst., 20(5), May 2021, DOI: 10.1145/3458768

List of abbreviations

AGV	Automated Guided Vehicle
BW	Bandwidth
CNC	Centralized Network Controller
CoAP	Constrained Application Protocol
DetNet	Deterministic Networking
DGM	Disjunctive Graph Model
FIFO	First in, first out
GCD	Greatest Common Divisor
GCL	Gate Control List
JSON	Java Script Object Notation
KPI	Key Performance Indicator
NETCONF	Network Configuration Protocol
PD	Packet Delay
PDC	Packet Delay Correction
PDV	Packet Delay Variation
PLC	Programmable Logic Controller
PSFP	Per-Stream Filtering and Policing
PTP	Precision Time Protocol (IEEE 1588)
RPC	Remote Procedure Call
QoS	Quality of Service
RESTCONF	Representational State Transfer Configuration
SOAP	Simple Object Access Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TSC	Time-Sensitive Communication
TSN	Time Sensitive Networking
VLAN	Virtual LAN
XML	eXtensible Markup Language
YANG	Yet Another Next Generation

Table 1: List of abbreviations

Terms and Definitions

6GDetCom node	A wireless TSN bridge using 6G technology for implementing the wireless transmission.
Asynchronous Traffic Shaper	A packet scheduling mechanism defined in the TSN standard IEEE 802.1Qcr.
Augmented Reality	Presentation concept that combines physical (real) world and computer-generated content.
Automated guided vehicle	A vehicle automatically following a given path using sensors, e.g., on a shop floor in a factory.
Automatic repeat request	Protocol to control errors by automatically repeating erroneous and dropped frames.
Bridge	A layer 2 network element forwarding packets between network segments in an Ethernet network.
Bridge delay	See port-to-port delay.
Centralized Network Controller	An entity of the network control plane that controls network elements like bridges based on a global view onto the network.
Control plane	Part of a network responsible for controlling the functions of the network data plane, e.g., configuration of forwarding tables.
Cyber Physical System	A system integrating (virtual) compute resources and physical entities like physical machines on a shop floor or other physical objects.
Data plane	Part of a network responsible for transporting data between source and destination (packet forwarding).
Dependability	In the context of real-time communication, dependability refers to the ability of a system or software to consistently deliver the expected functionality and performance while ensuring its correctness and reliability. It encompasses several key attributes that are crucial for real-time systems, including availability, reliability (see below), safety, fault tolerance, timeliness, and predictability.
Deterministic Networking	A standardization effort by the Internet Engineering Taskforce (IETF) to enable communication with bounded loss, packet delay, packet delay variation over layer 3 (routed) networks.
Earliest Deadline First	Real-time scheduling strategy always selecting the queued item with the earliest deadline for execution or forwarding.
Edge cloud	A compute (server) infrastructure located close to the stations executing applications to reduce latency between applications and services hosted in the infrastructure and to increase the performance and efficiency of clients, e.g., by offloading resource-intensive tasks to the edge cloud servers.
EtherCat	Real-time Ethernet technology.
Extended Reality	Collective term for Augmented Reality (AR), Virtual Reality (VR), and Mixed Reality (MR).
FIFO	Scheduling strategy selecting the queued item next that has been waiting the longest for execution or forwarding.

Frame preemption	A network mechanism to interrupt (preempt) lower-priority frames in transmission by higher-priority frames and later resume the transmission of the lower-priority frame, defined in TSN standard IEEE 802.1Qbu.
Frame replication	Sending multiple copies (replicas) of a frame over different network paths to increase reliability, defined in TSN standard IEEE 802.1CB.
Gate	Mechanism in a TSN bridge to define which egress queues of a port are eligible to forward packets.
Gate Control List	Table defining when to open and close gates at a TSN bridge according to a scheduling table.
Graceful degradation	In the context of end-to-end scheduling: capability to maintain reduced guarantees with respect to timeliness and reliability instead of steeply dropping to no guarantees.
Jitter	Variation in packet delay.
Listener	TSN term for receiver or destination of packets.
Mixed Reality	Systems combining physical (real) objects and computer-generated objects.
Offloading	Executing an application on a remote machine, often used with resource-poor mobile devices that offload resource-intensive tasks to a server infrastructure to increase efficiency.
Packet Delay	Delay of a packet between two reference points such as starting the transmission of a packet at the source station and receiving the packet at a network element or end station (destination).
Packet Delay Variation	Packet delay variation describes the amount of variation of the latencies perceived when a series of messages is transmitted from a given sender to a given receiver over a given network.
Packet Scheduling	Function of the network data plane to decide when to forward queued packets by bridges or routers.
Per-Stream Filtering and Policing	Mechanisms defined in the TSN standard IEEE 802.1Qci for frame counting, filtering, policing, and service class selection.
Port-to-port delay	The delay that a packet experiences between the ingress port (packet received by bridge) and egress port (packet transmitted bridge) of a bridge.
Powerlink	Real-time Ethernet technology.
Precision Time Protocol	A network protocol and mechanisms to synchronize clocks at different devices in a networked system, defined in the standard IEEE 802.1AS.
Priority Code Point	The priority code point is a header field defining the priority of an Ethernet frame by a number in the range from 0 to 7.
PROFINET	A real-time Ethernet technology.
Programmable Logic Controller	Device (computer) to automatically control a manufacturing process, e.g., robots or machines on a shop floor.
Radio Access Network	A part of a mobile communication network providing access of end stations like mobile phones to the core network using a radio technology.
Reliability	Reliability describes the probability that a system will meet its expected performance metrics and perform its intended functions consistently and correctly over time.

Robust schedule	A schedule that provides guarantees for uncertain parameters.
Software-Defined Networking	An approach for network management based on concepts such as logically centralized network management.
Southbound interface	Interface between Centralized Network Controller and bridges in the network control plane.
Talker	TSN term for sender or source of packets.
Time-Aware Shaper	A packet scheduling mechanism for traffic according to the TSN standard IEEE 802.1Qbv (time-driven scheduling).
Time-Sensitive Networking	Collective term for a set of standards by the Institute of Electrical and Electronics Engineering (IEEE) for real-time communication over IEEE 802 networks.
Virtual LAN	A logical (virtual) local-area network implemented atop a physical layer 2 network (Ethernet).
Virtual Reality	Presentation concept that presents the user with computer generated three-dimensional content.
Wireless by-pass	By-passing several wireless links by a wireless network.

Table 2: Terms and Definitions

5 Appendix

5.1 YANG Data Model

The full YANG data model for modelling stochastic PD as histograms is shown below. The YANG module can also be downloaded from Github[6].

```
module port-to-port-delay {
    yang-version "1.1";

    namespace urn:det6g:port-to-port-delay;

    prefix det6g;

    import ieee802-types {
        prefix ieee802;
    }

    import ieee802-dot1q-types {
        prefix dot1qtypes;
    }

    import ieee802-dot1q-bridge {
        prefix dot1q-bridge;
    }

    organization DETERMINISTIC6G;

    contact "https://deterministic6g.eu/";
    description
```

[6] https://github.com/DETERMINISTIC6G/deterministic6g_yang_models/

```
        "A YANG model for modelling stochastic port-to-port delay.";

    revision "2024-05-21" {
        description "Initial revision";
         reference "Deliverable D3.4 of DETERMINISTIC6G project";
    }

    grouping delay-histogram {
        description
             "Delay histogram";
        leaf start {
             type uint64;
             description
                  "The start value of the first bin in nano-seconds.
                  If not specified, the first bin starts at 0.";
        }
     leaf bin-count {
         type uint32;
         mandatory true;
         description
             "Number of bins.";
     }
     list bin {
         description
             "Bins of histogram.";
         key index;
         leaf index {
             type uint32;
         mandatory true;
         description
              "The index of this bin.";
             }
         leaf width {
             type uint64;
         mandatory true;
                 description
                  "The width of this bin in nano-seconds.";
             }
         leaf count {
                 type uint32;
         mandatory true;
             description
              "Count of values in this bin.";
             }
     }
     leaf tail {
             type uint32;
         description
             "Count of values in the tail of the histogram
              after the upper bound of last bin until infinity.
              Can be used to define an unbounded distribution.";
        }
    }
```

```
    augment "/dot1q-bridge:bridges/dot1q-bridge:bridge/dot1q-
bridge:component" {
        container port-to-port-delays {
            config false;
            list port-to-port-delay {
                key "ingress-port egress-port traffic-class index";
                leaf ingress-port {
                    type dot1qtypes:port-number-type;
                    config false;
                    mandatory true;
                    description
                        "Unique number of ingress port.";
                }
                leaf egress-port {
                    type dot1qtypes:port-number-type;
                    config false;
                    mandatory true;
                    description
                        "Unique number of egress port.";
                }
                leaf traffic-class {
                    type dot1qtypes:traffic-class-type;
                    config false;
                    mandatory true;
                    description
                        "Traffic class (0..7)";
                }
                leaf index {
                    type uint16;
                    config false;
                    mandatory true;
                    description
                        "Index to define multiple histograms per
                         port-pair and traffic class.";
                }
                uses delay-histogram;
            }
            leaf dependency-class {
                type enumeration {
                    enum "independent";
                    enum "dependent";
                }
                description
                    "Are the given delays only applicable under
                     certain conditions (e.g., for frames of
                     certain length)?";
            }
            container validity-period {
                container valid-from {
                    description
                        "Given delays are only valid at or
                         after this point in time, specified as
```

```
                        PTP timestamp.";
                    uses ieee802:ptp-time-grouping;
                }
                 container valid-until {
                    description
                        "Given delays are only valid until
                         this point in time, specified as PTP
                         timestamp.";
                    uses ieee802:ptp-time-grouping;
                 }
             }
         }
     }

    augment "/dot1q-bridge:bridges/dot1q-bridge:bridge/dot1q-
bridge:component/port-to-port-delays" {
        when "dependency-class = 'dependent'";
         container dependencies {
            container length-dependency {
                leaf min-frame-length {
                    type uint32;
                    description
                        "Values apply only to frames equal or
                         greater than this value.";
                 }
                 leaf max-frame-length {
                    type uint32;
                    description
                        "Values apply only to frames equal or
                         smaller than this value.";
                 }
            }
         }
    }
}
```