DETERMINISTIC6G

Digest on First DetCom Simulator Framework Release

D4.1

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

Digest on First DetCom Simulator Framework Release

Grant agreement number:	101096504
Project title:	Deterministic E2E communication with 6G
Project acronym:	DETERMINISTIC6G
Project website:	Deterministic6g.eu
Programme:	EU JU SNS Phase 1
Deliverable type:	Public Software Release
Deliverable reference number:	D4.1
Contributing workpackages:	WP4
Dissemination level:	PUBLIC
Due date:	31-12-2023
Actual submission date:	21-12-2023
Responsible organization:	USTUTT
Editor(s):	Frank Dürr and Lucas Haug
Version number:	v1.0
Status:	Final Version
Short abstract:	This deliverable describes the simulator framework, which is used in the project to validate the concepts that are developed in the project. It provides an overview of the features so far implemented for the simulator and describes the design rationales behind these implementations. The core feature of the first release is the simulation of packet delay variation in the data plane of a TSN network and processing delay variation of applications and services hosted on an edge cloud infrastructure. Besides a description of these features, also an overview of already available delay distributions is given, which are released together with the software. Finally, we also present the design and implementation of validation scenarios for the simulator framework, which are used, for instance, to validate and evaluate concepts for wireless-friendly end-to-end scheduling and the performance of time synchronization under PDV.
Keywords:	5G, 6G, software, validation, simulator, packet delay, processing delay, packet delay variation, dependable communication, TSN, PTP

Contributor(s):	Joachim Sachs (EDD) Junaid Ansari (EDD) Mahin Ahmed (SAL) Damir Hamidovic (SAL) Muzaffar Raheeb (SAL) Jose Costa-Requena (CMC) James Gross (KTH) Gourav Prateek Sharma (KTH) Frank Dürr (USTUTT)

	Lucas Haug (USTUTT) János Harmatos (ETH) Marilet De Andrade Jardim (EAB) Oliver Höftberger (B&R) Franz Profelt (B&R) David Puffer (B&R)

Revision History

06/11/2023	Draft version for first internal review
11/12/2023	Draft version for second internal review
19/12/2023	Final version

Disclaimer

This work has been performed in the framework of the Horizon Europe project DETERMINISTIC6G co-funded by the EU. This information reflects the consortium's view, but the consortium is not liable for any use that may be made of any of the information contained therein. This deliverable has been submitted to the EU commission, but it has not been reviewed and it has not been accepted by the EU commission yet.

Executive summary

This digest gives an overview of the open-source software released as deliverable D4.1 *DetCom Simulator Framework Release 1*. It describes the main features of the simulator software, which is based on the OMNeT++[1]/INET simulation framework[2] and the design rationales behind these features. It is not meant as a programmer's or user's guide, but provides pointers to the open-source code repository found at Github and (as snapshot) at the Zenodo platform (see links in Table 1).

The core features implemented for the first software release of the simulator are simulation models to simulate the characteristic stochastic Packet Delay (PD) in the network data plane – in particular, for 6GDetCom nodes implementing wireless Time Sensitive Networking (TSN) bridges – and Processing Delay for applications and services hosted in an edge cloud computing infrastructure. Large Packet Delay Variation (PDV) – larger than in wired network infrastructures such as wired TSN networks – is a typical property of wireless networks, which directly impacts real-time communication mechanisms. In particular, time-driven scheduling mechanisms such as the Time-Aware Shaper (IEEE 802.1Qbv [IEEE15-8021Qbv]) are directly impacted by PDV. Therefore, novel algorithms to calculate robust schedules or novel scheduling mechanism are required that are able to deal with stochastic delay distributions. These novel scheduling approaches then need to be validated with respect to their ability to deal with stochastic delay and compared against existing mechanisms. Also processing delay may vary significantly in a virtualized cloud-hosted environment executing software in virtual machines or containers. Therefore, being able to simulate these delays is essential for validating the performance – e.g., in terms of max. latency guarantees or number of streams that can be accommodated in the network – of such mechanisms under delay variation and for developing the concepts proposed in this project to improve this performance. The approach chosen for simulating PD distributions and processing delay is generic in the sense that any delay distribution can be simulated through specifying closed formulas (probability distributions), directly integrating data sets (histograms), or algorithmically by implementing stochastic processes, allowing also for correlated delay. In particular, data sets measured in a 5G testbed as part of the project can be integrated. The simulation models for PD have been integrated into the existing INET simulation models for TSN bridges; processing delay simulation has been integrated into simulated application components in INET. Exemplary PD and processing delay data sets for wired and wireless bridges and edge cloud applications are also briefly introduced in this document and released together with the software.

Moreover, an initial set of validation scenarios implemented for the simulator are described in this document, including a baseline scenario, an industrial use case scenario, and a scenario for evaluating time synchronization under PDV.

[1] OMNeT++ Discrete Event Simulator (omnetpp.org)
[2] INET Framework - INET Framework (omnetpp.org)

Contents

1 Introduction

This document gives an overview of the simulator software released as deliverable D4.1 *DetCom Simulator Framework Release 1*. Note that the type of D4.1 is an open-source software (not primarily a report). This digest accompanies the software release to provide an easy-to-read overview of the features implemented in D4.1, their general design, and the design rationales. This helps readers who are not proficient in software development and the simulator framework to understand the contributions of the deliverable. It is not meant as programmer's guide or user's guide and does, therefore, not provide exhaustive documentation of the software, nor instructions how to use this software. The essential content of the deliverable is the software (code) and the software artefacts (software documentation including class and showcase descriptions, configuration files, executable container images, etc.), which can be found at the project's public Github repository and as a snapshot of this repository at the Zenodo platform. Links to the software and data sets are provided in Table 1.

Component name		License	Link
Simulator Framework		LGPL v3	GitHub: DETERMINISTIC6G/deterministic6g Zenodo DOI: 10.5281/zenodo.10401977
PD Datasets (see Sec. 2.2.3)	PD-Wired	CC BY-SA 4.0	GitHub: deterministic6g_data/PD-Wired Zenodo DOI: 10.5281/zenodo.10405085
	PD-Wireless-5G-1	CC BY 4.0	GitHub: deterministic6g_data/PD-Wireless-5G-1 Zenodo DOI: 10.5281/zenodo.10390211
	PD-Wireless-5G-2a	CC BY-ND 4.0	GitHub: deterministic6g_data/PD-Wireless-5G-2a Zenodo DOI: 10.5281/zenodo.10405085
	PD-Wireless-5G-3a	CC BY-ND 4.0	GitHub: deterministic6g_data/PD-Wireless-5G-3a Zenodo DOI: 10.5281/zenodo.10405085
Processing Delay Datasets (see Sec. 4.3)	ProcessingDelay Distribution1	CC BY-ND 4.0	GitHub: deterministic6g_data/ProcessingDelayDistribution1 Zenodo DOI: 10.5281/zenodo.10405085
	ProcessingDelay Distribution2	CC BY-ND 4.0	GitHub: deterministic6g_data/ProcessingDelayDistribution2 Zenodo DOI: 10.5281/zenodo.10405085

Table 1: Overview over subcomponents of this deliverable.

This digest provides, where appropriate, links to the software (code), software documentation, and instructions (e.g., README files) that are part of the repository to facilitate navigation to the relevant parts of the repository. More detailed technical information about the implementation can be found at these places in the repository.

The software described here is based on the OMNeT++ simulator [OMN23] and the INET framework [INE23] for network simulation, which are both available as open-source software. Both come with extensive documentation, which is not replicated here. However, where necessary, we provide sufficient information to keep this document self-contained and refer to the detailed information available on the OMNeT++/INET web pages.

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 validation framework. Readers who already know the DETERMINISTIC6G project could skip this sub-section and directly start with Section 1.2 motivating the need for the validation framework and describing its purpose. Afterwards, we described the relation to other work packages of the project, before giving an overview of the remainder of this document.

1.1 Scope of the Project: The 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 % [3GPP16-22261]. 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 KPI (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 Controller (PLC), and is based on a wired

communication system, such as EtherCat and Powerlink, which is limited to local and isolated network domains which is configured to the specific purpose of the local applications. With the standardization of Time-Sensitive Networking (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.

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 on top 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, we extend a notion of deterministic communication (where all behavior of network and compute nodes and applications is pre-determined) 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.

It is critical to validate the effectiveness and efficiency of the concept for dependable time-critical communication mentioned above. The validation framework presented in this report serves as the main tool for this validation. Next, we describe this purpose of the validation framework within the DETERMINSTIC6G project in more detail.

1.2 Purpose of the Validation Framework

The primary purpose of the validation framework is to facilitate the validation of the concepts and mechanisms developed as part of other work packages of the DETERMINISTIC6G project through simulations. The novel concepts and mechanisms of the DETERMINISTIC6G project mentioned above are translated to simulation models, which are then used in scenarios based on use cases of time-critical communication for a quantitative evaluation.

In particular, the characteristic stochastic delay of communication over wireless links is simulated and its impact onto time-dependent mechanisms such as time-driven packet scheduling and time synchronization is evaluated. To realistically simulate network delay, we follow a data-driven approach, where delay models are based on measurements from real system such as a 5G experimental testbed. Moreover, we also consider the delay induced by edge cloud components to assess the mentioned end-to-end KPIs, which include network delay as well as processing delay “on the edge”.

1.3 Relation to other Work Packages

As already mentioned, the simulation models of the validation framework developed in WP4 are based on the concepts and mechanisms WP 2 *6G Centric Enablers for Deterministic Communication* and WP 3 *Enablers of 6G Convergence for Deterministic Communication*; the implemented validation scenarios are related to the use cases defined in WP 1 *Vision, Architecture and System aspects for Deterministic E2E Communication with 6G*. Figure 1 depicts the embedding of WP4 into the project.

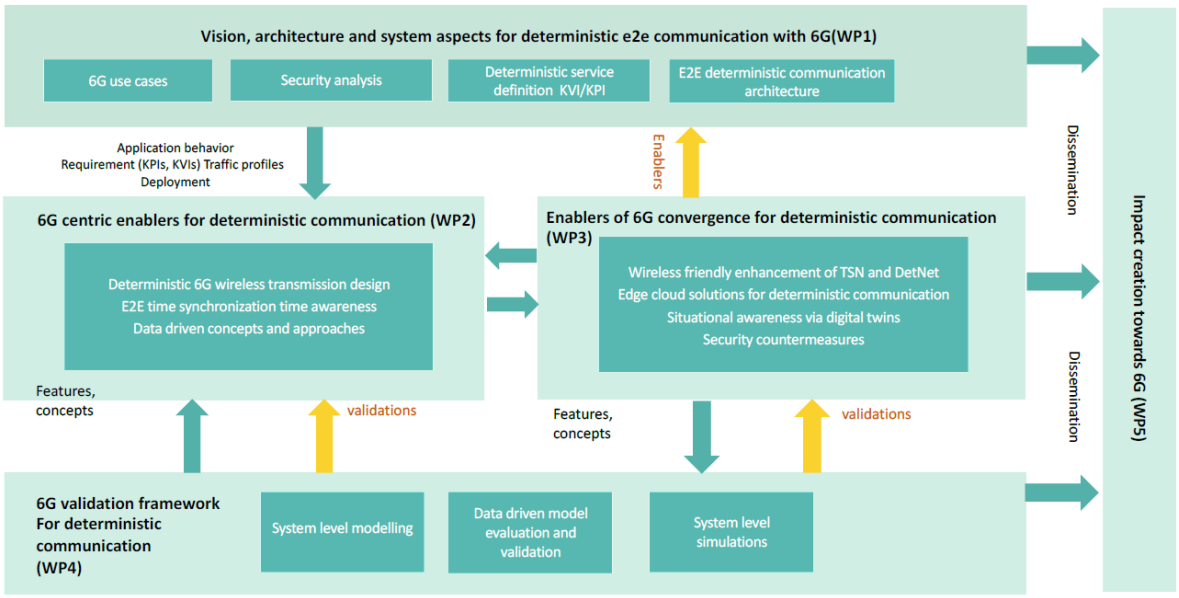

Figure 1: Relation of WP4 to other work packages

WP1, WP2, and WP3 have also released reports describing the concepts behind the implementation in more detail. Therefore, although this report tries to provide sufficient information to keep the document self-contained and readable, we refer to the reports of the other deliverables for in-depth information about the fundamental concepts and use cases implemented here:

- D1.1: DETERMINISTIC6G Use Cases and Architecture Principles [DET23-11]
- D2.1: 6G Centric Enablers [DET23-21]
- D2.2: Time Synchronization for E2E Time Awareness [DET23-22]
- D3.1: 6G Convergence Enablers Towards Deterministic Communication Standards [DET23-31]

Please also note that, although the purpose of the software is to validate the concepts developed in the project, the validation results are subject to another upcoming deliverable D4.5 *Validation Results*. Thus, validation results presented as part of this report are preliminary.

Moreover, a second release of the simulator framework will follow at the end of the project, thus, the design of the first release will be extended and possibly adapted at some places, where necessary.

1.4 Structure of the Document

The remainder of this digest is structured as follows:

In Section 2, we present simulation models for the network data plane where the forwarding of packets takes place. Here, we focus on the essential concept for simulating wireless 6G networks together with wired TSN networking infrastructures: PD distributions with high PDV. PDV has direct impact onto time-driven scheduling mechanisms such as the Time-Aware Shaper (TAS) defined in IEEE 802.1Qbv [IEEE15-8021Qbv] or time synchronization. The realistic simulation of PD distributions allows for the evaluation of this impact, and, therefore, is the central concept of the simulation framework with respect to the realistic simulation of the wired/wireless data plane.

In Section 3, we present the current implementation of the network control plane used to configure data plane mechanisms in the simulator. Since the focus of the first release is on the data plane mechanisms, only a rudimentary control plane implementation is presented here.

In Section 4, we focus on simulation models for the edge cloud infrastructure. In concert with the PDV concept for packet delay distributions, we also model the delay of edge cloud components as processing delay distributions in the simulator.

In Section 5, we describe the implementation of several validation scenarios for the simulator, which use the other concepts such as simulation of PD and processing delay. Currently, these scenarios aim at evaluating the impact of PDV onto end-to-end scheduling and the performance of time synchronization.

Finally, we conclude this report in Section 6 with a brief summary and outlook onto future work.

2 Simulation Models for the Network Data Plane

In this section, we describe the essential simulation models of the network data plane. The network data plane is responsible for forwarding packets according to the configuration made by the network control plane in terms of where to forward packets (e.g., over which egress port of a bridge), and when to forward packets (packet scheduling or shaping in TSN terminology). To this end, the control plane is configuring the forwarding tables and scheduling mechanisms, such as the gate control lists (timetables) of time-aware egress ports of TSN bridges, and the data plane is executing the forwarding and scheduling as configured by the control plane.

The data plane can contain wired and wireless network elements. The simulation framework is focused on dependable time-sensitive communication in TSN, i.e., the typical network elements are wired and wireless TSN bridges. We also call wireless TSN bridges 6GDetCom[3] nodes, which are the major novel components in our simulation framework since they encompass wireless links with very specific timing characteristics that are fundamentally different from wired TSN bridges. Since most end-to-end communication that we consider in this project passes through at least one 6GDetCom node at some point along the end-to-end path, the design and implementation of 6GDetCom nodes is the first and foremost component in our first software release, and, therefore, will be described in more detail in Section 2.2, after briefly introducing the already existing basis for this component: the simulation model of the standard wired TSN bridge (Section 2.1).

2.1 Starting Point: Existing TSN Simulation Models

The 6GDetCom node is an extension of the simulation model of a TSN bridge implemented by the INET framework. Besides the standard forwarding mechanisms of an IEEE 802.1Q bridge, the 6GDetCom node inherits the TSN shaping mechanisms for traffic scheduling. To keep this document self-contained, we first briefly summarize the TSN data plane mechanisms already implemented by the INET framework, before we present the specific extensions of the 6GDetCom node for wireless communication.

[3] In this document, we always use the term 6G when referring to mobile networks, although some concepts and implementations might also be applicable to existing 5G networks.

INET offers a rich set of TSN standard implementations in version 4.5.2 (at the time of publishing this document), including but not limited to:

- IEEE 802.1Qbv Enhancements for Scheduled Traffic [IEEE15-8021Qbv]: time-aware shaping with gating mechanism at egress queues
- IEEE 802.1Qav Forwarding and Queuing Enhancements for Time-Sensitive Streams [IEEE09-8021Qav]: credit-based shaping with send slopes and idle slopes
- IEEE 802.1Qcr Asynchronous Traffic Shaping [IEEE20-8021Qcr]: asynchronous traffic shaping with per-class queuing and per-stream reshaping
- IEEE 802.1Qci Per-Stream Filtering and Policing [IEEE17-8021Qci]: per stream filtering and policing
- IEEE 802.1CB Frame Replication and Elimination for Reliability [IEEE17-8021CB]: frame replication and elimination mechanism to send replicated frames over different paths to increase reliability
- IEEE 802.1Qbu Frame Preemption [IEEE16-8021Qbu]: preemption of low-priority frames in transmission by high-priority (express) frames
- IEEE 802.1AS Timing and Synchronization for Time-Sensitive Applications [IEEE11-8021AS]: gPTP for time synchronization; drifting oscillators; multiple clocks for multiple time domains

In the first release, we utilize the TAS (IEEE 802.1Qbv Enhancements for Scheduled Traffic) implementing a time-driven scheduling mechanism, since such time-driven mechanisms are directly impacted by PDV. The TAS relies on timetables (so-called gate control lists (GCL)) driving so-called gates behind each egress queue of TSN bridges. If a gate is open, frames from this queue might be transmit; if a gate is closed, the queue is not eligible for transmission. GCLs are configured by the so-called Centralized Network Controller (CNC) (in the fully centralized TSN model) to open and close gates at specific points in time according to a schedule. To calculate such schedules automatically, specific algorithms are used that consider the timing constraints of streams such as their bounds on end-to-end network delay. This time-driven mechanism has obvious dependencies on PDV, making it a primary target, for instance to evaluate robust time-driven schedules that can deal with PDV, although later we might also consider other shaping mechanisms.

Figure 2 shows the internal architecture of a TSN bridge with two egress queues as implemented by INET in the simulator. Most interesting with respect to the TAS is the MAC layer of the Ethernet interface (cf., *eth[0]* in the green box at the bottom left of the figure). The internal components of the MAC layer (yellow box) show the egress queue component of the Ethernet interface, whose internal components are shown in the blue box. In the blue box, we can see the individual egress queues for the interface as well as the classifier, which enqueues packets in the queue corresponding to the packet's priority to traffic class mapping. The figure also shows the transmission gates of the TAS behind each queue.

We refine this architecture in Section 2.2.2 to add the characteristic PD distributions of wireless 6GDetCom nodes to the TSN data path.

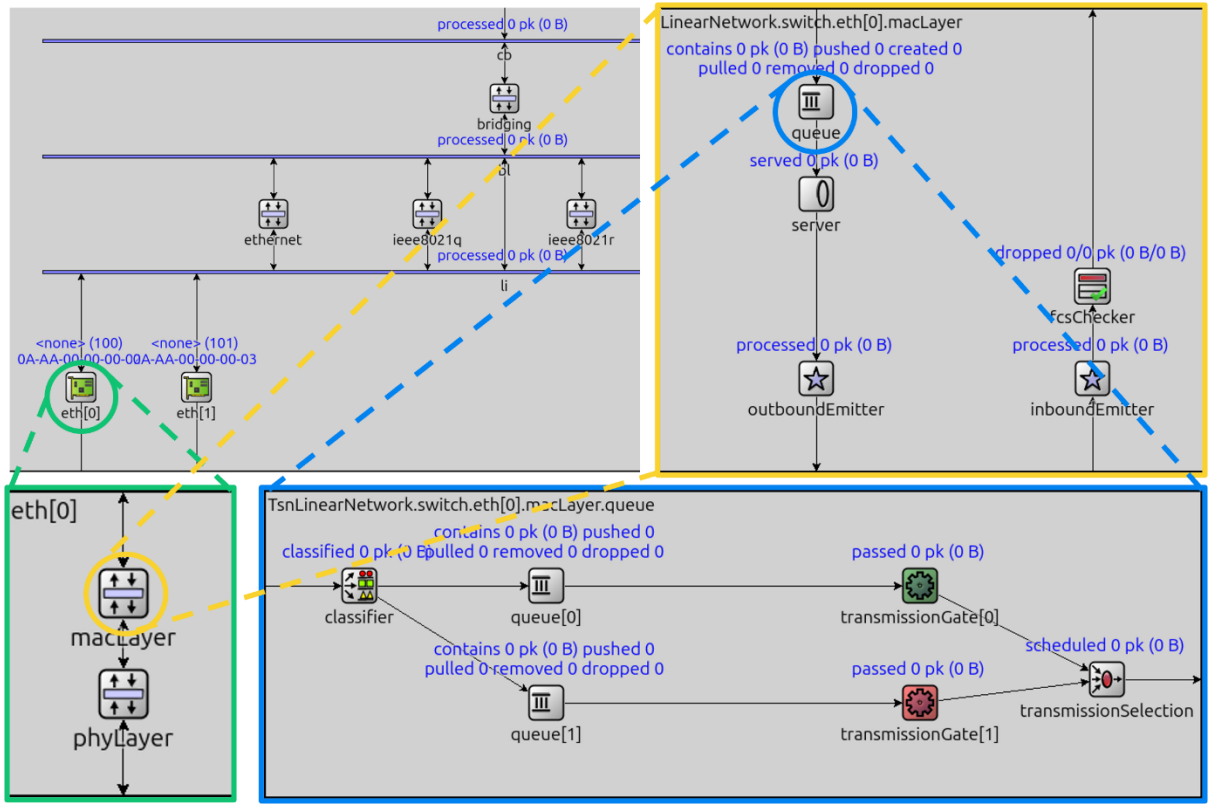

Figure 2: Internal architecture of a TSN bridge in INET.

2.2 Simulating Packet Delay Variation in the TSN Data Plane

The delay of a packet while passing through a 6GDetCom node from ingress to egress port is qualitatively and quantitatively different from the delay of a wired TSN bridge. Therefore, the accurate simulation of the characteristic PD of wired and wireless TSN bridges is essential for evaluating the performance of TSN shaping mechanisms, algorithms for calculating "wireless-friendly" IEEE 802.1Qbv schedules (cf. Section 4 of D3.1: *6G Convergence Enablers Towards Deterministic Communication Standards* [DET23-D31]), time synchronization mechanisms (cf. D2.2: *Time Synchronization for E2E Time Awareness* [DET23-D22]), etc.

Next, we present our simulation models for simulating stochastic PD in 6GDetCom nodes.

2.2.1 Alternative Approaches and Design Rationale

First of all, simulation models are, according to the model theory of Stachowiak [S73], pragmatic abstractions of another object. Abstraction means that the model will not simulate all functions and properties of the abstracted object. Pragmatic means that the model must – despite being an abstraction – be fit for a given purpose, in our case, for validating the developed concepts and evaluating performance realistically. Obviously, there is no single approach to model a 6GDetCom node. Next, we compare two alternative approaches, select one, and explain the design rationale of our decision.

With respect to functional requirements, our 6GDetCom node must simulate the TSN mechanisms already listed in Section 2.1 since 5G and 6G nodes should at their "boundaries" behave like a wireless TSN bridge (cf. architecture described in [DET23-D11]).

With respect to non-functional requirements, packets inside the 6GDetCom node should be delayed with a characteristic delay distribution.

Two fundamental approaches exist to tackle the second requirement of simulating characteristic PDs:

1. **A fine-grained simulation model** of all layers of the network stack and influencing factors of the environment. In particular, this includes the physical layer and environment, which leads to effects like shadowing, slow and fast fading, diffraction, interference, but also protocol details such as multiplexed media access, automatic repeat requests, etc.
2. **A coarse-grained simulation model integrating stochastic delay distributions** in a form that can be evaluated by the simulator at simulation time to pick delays for packets to be forwarded from the given PD distributions.

The first approach is characterized by a low level of abstraction and consequently high complexity with respect to the very detailed implementation and simulation runtime. This problem can be alleviated by using an existing simulator for 5G systems such as Simu5G [NSS+20]. However, it is unclear (corresponding to a higher risk) (a) whether the resulting delay is accurate, i.e., whether the simulation model is indeed sufficiently fine-grained; (b) what the complexity of integrating various TSN mechanisms from INET is, which has not been a focus of Simu5G so far.

The second approach is characterized by a high level of abstraction – at least with respect to simulating wireless communication – and relies on realistic data such as measurements taken in real systems. The already reduced complexity of this approach can be further reduced by basing the implementation on the existing TSN-ready INET components (existing TSN bridge simulation model shown in Fig. 1).

We decided to implement the second approach using stochastic PD distributions in the simulator for the following reasons:

- Feasible complexity of implementation (feasible within the given project runtime).
- Availability of empirical data (Task 4.3 *Data-driven Analysis for RAN Latency Inference*).
- Higher flexibility by being able to “plug-in” any PD distribution (based on real measurements or artificial).
- Introducing and then simulating new wireless functions or enhancements is not the primary target. We rather simulate the effects on PD and the impact of PD onto TSN mechanisms and end-to-end performance.

The concrete delay distribution to be simulated is outside the scope of this deliverable, and subject to an upcoming deliverable D4.3 *Latency Measurement Data and Characterization of RAN Latency from Experimental Trials*. However, in this report we will show some examples or preliminary delay distributions and include them in the software repository.

Moreover, our implementation of stochastic packet delays is generic and extensible, enabling the specification of closed-form stochastic distributions (e.g., a normal distribution and its parameters), direct integration of data sets (histograms) stemming from measurements, or algorithms evaluated at simulation time to implement random processes (e.g., random walks). This allows for implementing independent and correlated packet delays, the parameters of which can also be dynamically adapted at simulation time using, for instance, the Scenario Manager feature of INET.

2.2.2 Design and Implementation of 6GDetComNode with Packet Delayer

The design of our 6GDetCom node is based on the wired TSN bridge component of INET such that it inherits all functions of a TSN bridge. The main missing part in the original TSN bridge design is the simulation of the characteristic stochastic PD of the wireless links of the 6GDetCom node when packets pass from UE to the RAN user plane (cf. Figure 3).

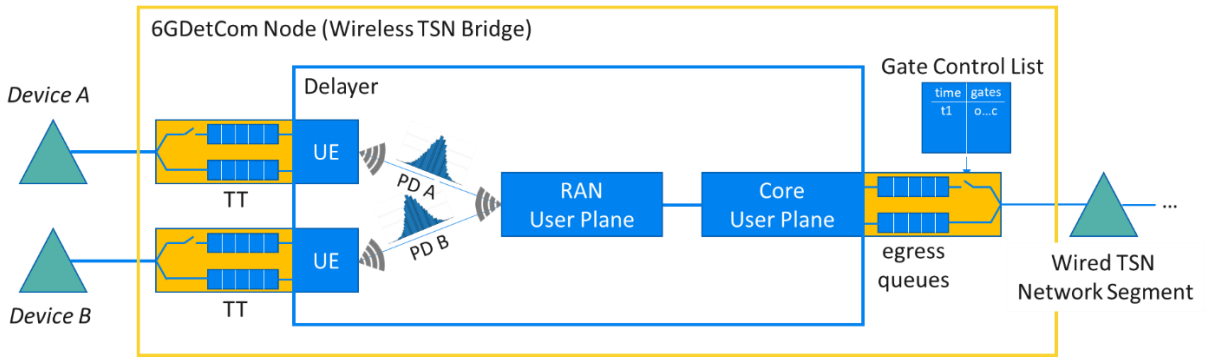

Figure 3: 6GDetCom node with two devices on the left connected through wireless links of a 6GDetCom node to wired TSN network segment on the right.

To add the characteristic PD, PDs are simulated by a subcomponent within the 6GDetCom node called *Delayer*. Note that the term Delayer is a term coined by the INET framework, which already provides hooks in its Ethernet bridge model implementation to add delay to packets passing through the bridge by adding Delayer components to the data path. Every packet that is transmitted from or to a UE passes through a Delayer component.

Figure 4 shows the structure of the 6GDetCom node based on the *TsnSwitch*[4] component of INET in more detail. Within the 6GDetCom node, the delay is added by a subcomponent called *PairwiseDelayer*[5](orange oval) within the so-called *bridging* layer (green circle). The *PairwiseDelayer* is a specific delayer component that can add individual delays per port pair and individually for upstream and downstream directions. Additionally, the 6GDetCom node implements a new UE interface type that allows for the transmission of packets without any additional simulated propagation and transmission delays since these are already part of the PD. In this module, packets arrive at one of the interfaces (i.e., eth[0], ue[0] or ue[1]) and traverse up to the *bridging* layer. This layer is responsible for defining the packet processing, e.g., identification of streams and ingress packet filtering. The *directionReverser* component then determines whether the packet shall be received by an internal application on the bridge (there is none in this example) or be forwarded to one or multiple egress interfaces. Within the *bridging* layer the packet then passes downwards through components, which for example determine the Priority Code Point (PCP) value and VID of the frame. It then reaches the proposed *PairwiseDelayer* component, which determines the delay to be

[4] TsnSwitch (omnetpp.org)
[5] PairwiseDelayer (deterministic6g.github.io)

added based on the incoming and outgoing interface. Finally, it arrives at its outgoing interface, which applies the configured TSN scheduling mechanisms, e.g., the TAS.

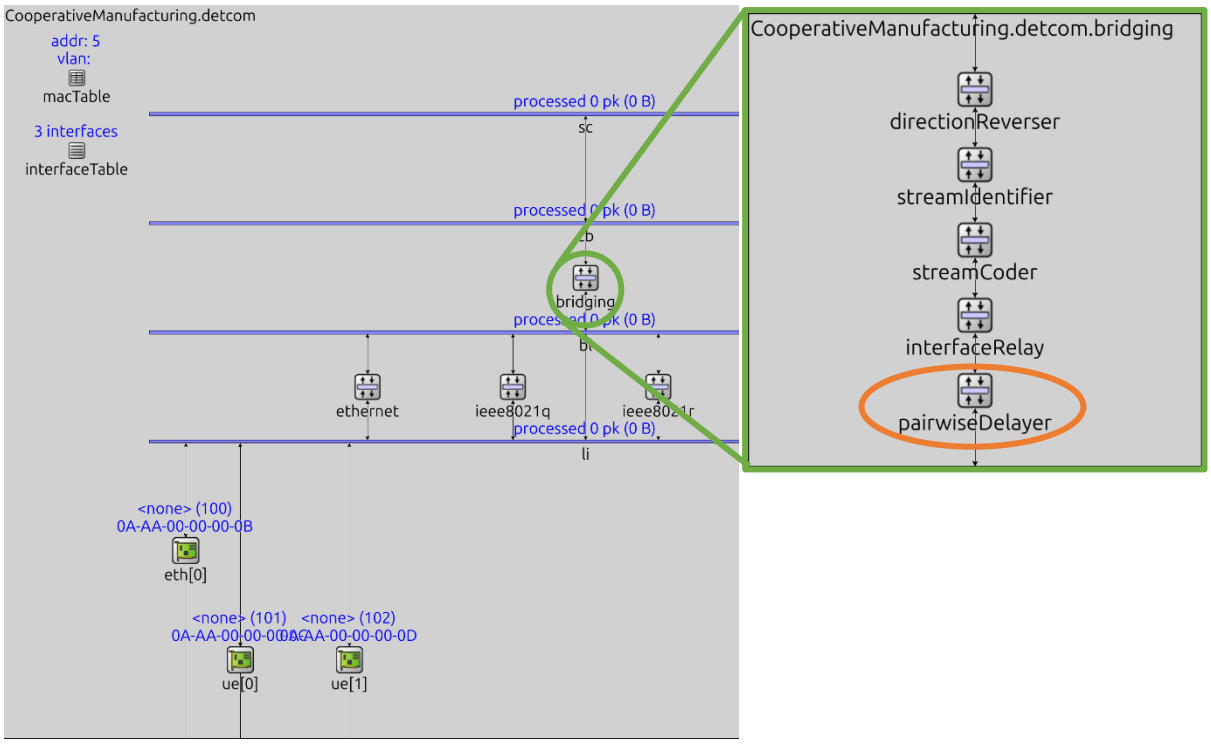

Figure 4: Structure of 6GDetCom node and the PairwiseDelayer component as part of the bridging layer.

PairwiseDelayer

In contrast to the capabilities of the standard INET delayer, the design of our *PairwiseDelayer*[5] component allows for specifying delays based on the incoming and outgoing ports of the 6GDetCom node, i.e., individually in upstream and downstream direction for any port pair. Listing 1 shows an example configuration file for the network depicted in Figure 3.

```
<delays>
    <delay in="deviceA" out="bridge">rngProvider("histogramUplink")</delay>
    <delay in="bridge" out="deviceA">rngProvider("histogramDownlink")</delay>
    <delay out="bridge">rngProvider("histogramFallback”)</delay>
    <delay>normal(5ms,2ms)</delay>
</delays>
```

Listing 1: Example PairwiseDelayer configuration file.

The definition of delays can either be a custom Network Description (NED) function (e.g., to implement a stochastic process or any other “algorithmic” definition of delay, or to pick delays based on a histogram data set), an OMNeT++ built-in statistical distribution (closed-form probability distribution), a mathematical expression (e.g., $1\ ms + 1\ ns$), or an arbitrary combination of the aforementioned options.

With the provided example configuration file, a packet sent from *deviceA* to *bridge* calls the provided *rngProvider()* function with *histogramUplink* as a parameter, which then returns a delay. The PairwiseDelayer allows to configure fallback options, e.g., a packet sent from *deviceB* to *bridge* would use the histogramFallback, while for any other packets the provided normal distribution applies.

The *rngProvider()* function accepts the identifier of another OMNeT++ component that implements our *IRandomNumberProvider* interface as function argument. In this first release of our simulation framework, this is the Histogram[6] module, as described in the next sub-section.

Histogram

OMNeT++ already provides built-in functions to pick random numbers from closed-form probability distributions, e.g., from a normal distribution or a uniform distribution. However, in some cases it might be desirable to simulate delays directly based on empirical data sets (measurements) available as histograms without fitting a function first. To this end, our simulation framework contains a Histogram[6] component, which can generate random numbers based on any kind of histogram in XML format. Listing 2 shows an example configuration file that corresponds to the histogram shown in Figure 5.

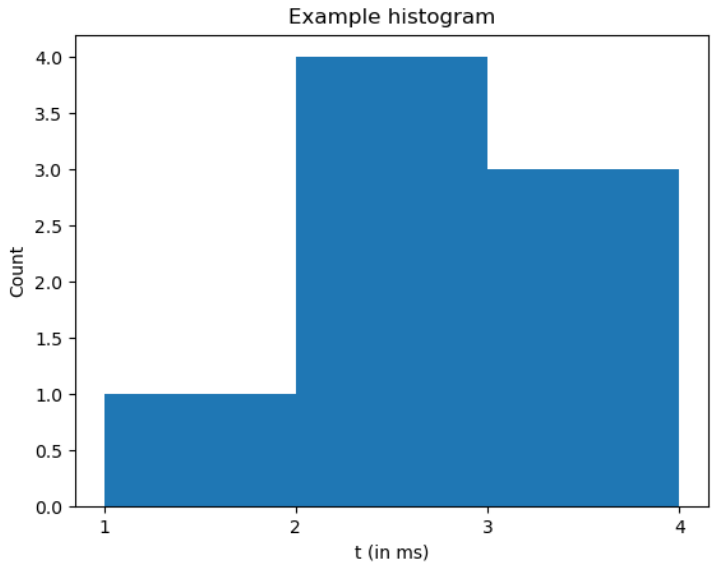

Figure 5: Example histogram

```
<histogram>
    <bin low="1ms">1</bin>
    <bin low="2ms">4</bin>
    <bin low="3ms">3</bin>
    <bin low="4ms">0</bin>
</histogram>
```

Listing 2: Example Histogram configuration file.

Each *bin* element covers a time interval ranging from value *low* to the *low* value of the next *bin* element and defines its value count as content of the element. The last element is only needed to define the upper bound of the last (previous) bin, and therefore, its count is always zero.

Note that these histograms can only model independent and identically (i.i.d.) distributed random delay variables. In particular, no correlation between delays can be modeled. Although we can model such correlated delays through stochastic process models, it is so far not possible in our framework to

[6] Histogram (deterministic6g.github.io)

directly integrate delay data while preserving the correlation between delays. This will be subject to future work.

2.2.3 Exemplary PD Models

The *PairwiseDelayer* requires closed-form formulas, data sets, or algorithms describing random processes to model PDVs. Please note that the following descriptions are not meant as detailed instructions for programmers on how to use the provided data sets in the simulator. Such instructions are provided as README files together with the data set in the repository. The following descriptions should provide information about what was measured and how to interpret the data. We provide the following PDVs as part of D4.1 (links to the datasets are provided in Table 1):

- PD-Wired (data set): PDV of a wired TSN bridge measured with a commercial TSN bridge. A description of how PD-Wired was measured can be found in the next section.
- PD-Wireless-5G-1 (data set): PD of wireless TSN bridge based on measurements from a 5G testbed as part of Task 4.3 *Data-driven Analysis for RAN Latency Inference*. Please note that this PD is preliminary. Final PDs will be part of deliverable D4.3 *Latency Measurement Data and Characterization of RAN Latency from Experimental Trials*, which is due later in month 24. Information on how PD-Wireless-5G-1 was measured is documented in [MSG23].
- PD-Wireless-5G-2a (data set): PD of a wireless TSN bridge based on measurements from a 5G testbed in an industrial research shopfloor, which were made in collaboration with the Fraunhofer Institute for Production Technology. The 5G testbed corresponds to a 5G standalone trial network with 100 MHz carrier bandwidth operating in the 3.7 GHz band. It is deployed at the 5G Industry Campus Europe in Aachen Germany[7]. The network setup and the measurement methodology are further described in [AAB+22]. Two histogram data sets are derived from measurements in the trial network, one measurement for downlink (from a controller to a mobile device over the 5G network, see Figure 6) and one measurement for uplink (from the mobile device over 5G to the controller, see Figure 7). The measurements are made for periodic PROFINET messages of 100 bytes size that are transmitted every 10 ms.

[7] https://5g-industry-campus.com/

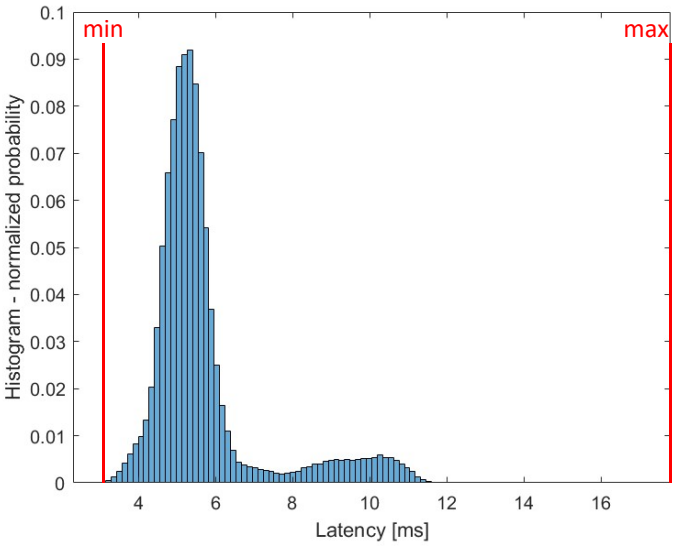

Figure 6: Histogram of downlink packet delay of the data set PD-Wireless-5G-2a based on measurements in a 5G trial network on a research production shopfloor.

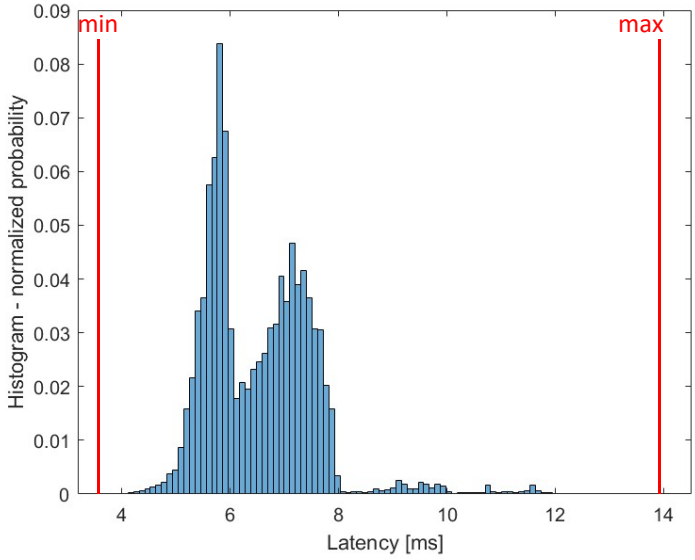

Figure 7: Histogram of uplink packet delay of the data set PD-Wireless-5G-2a based on measurements in a 5G trial network.

The data set is contributed with the objective to keep the quality of the data intact, which prohibits modification and further sharing of the data.

- PD-Wireless-5G-2b (closed formula): Closed form approximation of the data set PD-Wireless-5G-2a from a 5G trial network in an industrial research shopfloor as described above (see also [AAB+22]). The PDV is expressed as normal distributions that has been fitted to the histogram values
    - Downlink
        - Normal distribution (in units: ms)
        - Mean: $\mu = 5.69452$

        - Standard deviation: $\sigma = 1.5062$
    - Uplink
        - Normal distribution (in units: ms)
        - Mean: $\mu = 6.52078$
        - Standard deviation: $\sigma = 1.00279$
- PD-Wireless-5G-3a (data set): PD of a wireless TSN bridge based on measurements from a 5G testbed in an industrial research shopfloor, which were made in collaboration with the Fraunhofer Institute for Production Technology. The 5G testbed corresponds to a pre-commercial 5G URLLC standalone prototype network with 100 MHz carrier bandwidth operating in the 28 GHz band; it implements 5G standardized functionality for ultra-reliable and low latency communication [AVK+22]. The trial network is deployed at the 5G Industry Campus Europe in Aachen Germany[7]. The network setup and the measurement methodology are further described in [AAB+22] and in [AVK+22]. Two histogram data sets are derived from measurements in the trial network, one measurement for downlink (Figure 8) and one measurement for uplink (Figure 9). The measurements are made for periodic UDP messages with 32 bytes payload that are transmitted every 7 ms.

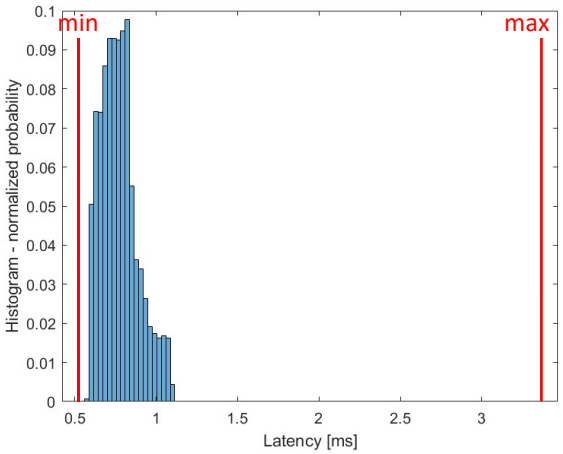

Figure 8: Histogram of downlink packet delay of the data set PD-Wireless-5G-3a based on measurements in a 5G URLLC trial network.

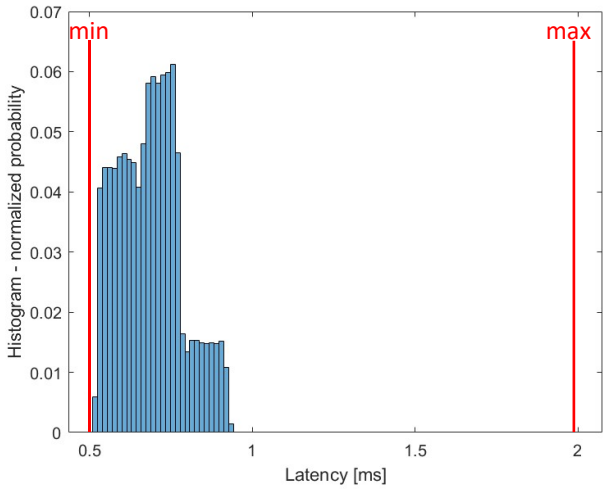

Figure 9: Histogram of uplink packet delay of the data set PD-Wireless-5G-3a based on measurements in a 5G URLLC trial network.

The data set is contributed with the objective to keep the quality of the data intact, which prohibits modification and further sharing of the data.

- PD-Wireless-5G-3b (closed formula): Closed form approximation of the data set PD-Wireless-5G-3a from a 5G trial network in an industrial research shopfloor as described above (see also [AAB+22] and [AVK+22]. The PD is expressed as normal distributions that has been fitted to the histogram values.
  - Downlink
    - Normal distribution (in units: ms)
    - Mean: $\mu = 0.776011$
    - Standard deviation: $\sigma = 0.122207$
  - Uplink
    - Normal distribution (in units: ms)
    - Mean: $\mu = 0.687132$
    - Standard deviation: $\sigma = 0.0976876$
- PD-Random-Walk (algorithm): an algorithmic definition of PDs for drawing correlated packet delays based on a random walk process. Please note that we use a random walk here only as an example to show that the simulation framework features the implementation of random processes in an algorithmic fashion and can also be used to produce correlated delay values at simulation time. We explicitly do not claim that random processes model PD in wireless networks! This is subject to ongoing investigations in the project, and results of these investigations will be presented at a later point in time in the project runtime The subsequent sections contain a description on how PD:Random-Walk is implemented.

PD-Wired

PD-Wired was measured with a commercial TSN bridge. Figure 10 shows the hardware setup of the measurements.

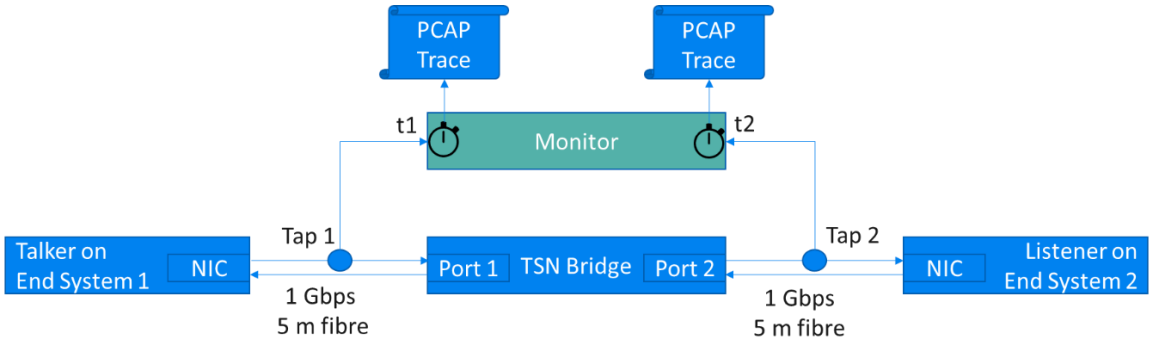

Figure 10: Setup for measuring PD of wired TSN bridge

The talker and listener are connected to the TSN bridge through fibre cables with 1 Gbps. Taps are installed on these cables to mirror the signal (packets) to two ports of a network monitoring device (Napatech NT40E3-4-PTP SmartNIC). These taps are passive devices, not adding extra delay. The monitoring device produces two trace files in Packet Capture (PCAP) format: one for the ingress traffic to the TSN bridge; one for the egress traffic from the TSN bridge. All captured packets are timestamped with nanosecond resolution, using the same hardware clock for both monitoring ports.

The bridge operates in store&forward mode. All gates of the TSN bridge are always open, i.e., no TSN shaping is performed. TSN shaping is used to control queuing delay and depends on the configuration of the gate states in the GCL. Here, we are only interested in the PD without queuing delay (i.e., only transmission, processing, and propagation delay, which are not controllable by the IEEE 802.1Qbv schedule), and open gates ensure that incoming packets are forwarded immediately. Therefore, besides the packets being measured, any other traffic is minimized (only control packets from the Address Resolution Protocol (ARP) and the Rapid Spanning Tree Protocol (RSTP) are sent by the bridge). VLAN filtering is turned on, as this is common in TSN networks since the tag is also required to carry the PCP header field for defining the priority of packets. Here, all packets have the same priority since TSN scheduling is not active.

A total number of 1000 minimum-sized (64B) measurement packets are sent with a rate of 10 Pkt/s, i.e., with a deliberately low rate such that no congestion is building up in the egress port queue.

Figure 11 shows the histogram of the packet delay between Tap 1 and Tap 2. The minimum delay is 4420 ns; the maximum delay is 4660 ns; the mean is with 95 % confidence in the interval [4545 ns, 4550 ns].

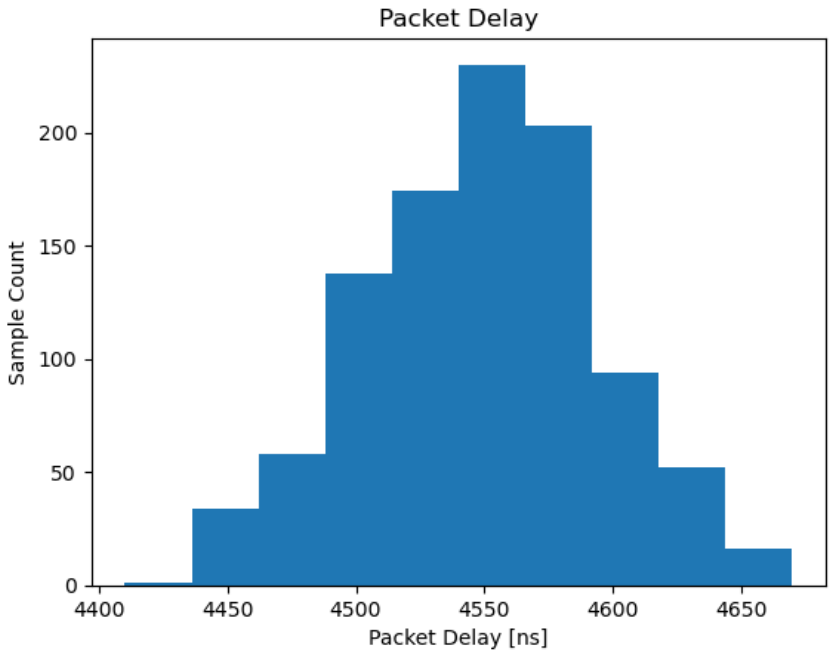

Figure 11: Packet delay histogram for wired TSN bridge

To interpret these measurements, it is interesting to further look at the contributors of the measured delay. Figure 12 shows that the measured delay actually includes transmission delay ($d_{transmission}$) for minimum sized packets over 1 Gbps links, propagation delay ($d_{propagation}$) along the cable (5 m), and processing delay ($d_{processing}$) of the TSN bridge. The first time stamp is taken at the tap at time t1 at the start of frame, the second at time t2 when the start of frame arrives at the second tap.

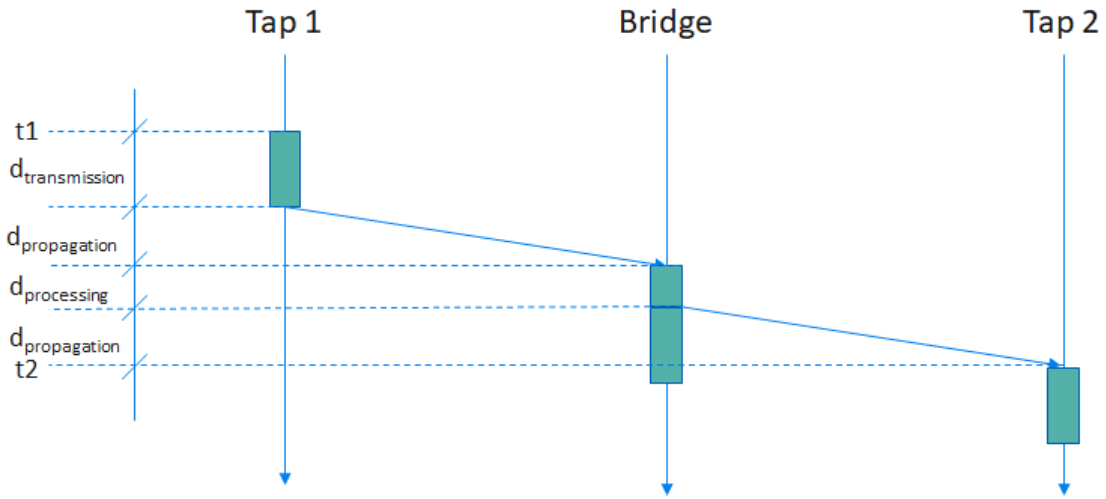

Figure 12: Break down of parts included in measured delay of wired TSN bridge

Therefore, the measured delay includes:

$$packet\ delay = 2d_{propagation} + d_{transmission} + d_{processing}$$

If one wants to separate the measured packet delay from transmission and propagation delay – which are often simulated already – to only consider the processing delay "inside" the bridge, the propagation delay and transmission delay can be estimated as:

$$d_{propagation} = \frac{5\,m}{\frac{2}{3}c} = 25 \text{ ns}$$

$$d_{transmission} = \frac{64B}{1\,Gbps} = 512\,ns$$

The mean value of the measured packet delay (including propagation, transmission, and processing delay) is 4546.4 ns. The estimated mean processing delay is therefore 4546.4 ns – 50 ns – 512 ns = 3984.4 ns.

PD-Random-Walk

A random walk is a random process that iteratively creates a sequence of correlated random numbers. We use random walks here as a deliberately simple and easy to understand example to show that we can implement PDVs through algorithms in our simulation framework. In the future, more sophisticated implementations can model, for instance, correlated PDs.

Given a random start value, say $x_0 \in \mathbb{R}$, in each iteration $i + 1$, a new random number is calculated as $x_{i+1} = x_i + r$, where r is a random number, e.g., ± 1.

In our simulation framework we implement this random walk process as a custom NED function shown in Listing 3.

```
quantity randomWalk(quantity init, quantity randValue, string key?)
```

Listing 3: Definition of the randomWalk NED function.

In this function, *init* represents the start value x_0 and *randValue* represents r. If multiple random walk processes are used within one simulation, the *key* parameter can be used to distinguish between them. The *quantity* type represents a numbered value including a unit (e.g. 5 ms). Listing 4 shows an example configuration using the *randomWalk* NED-function within our PairwiseDelayer.

```
<delays>
    <delay in="deviceA">randomWalk(5ms,normal(0ms,1ms),"uplinkA")</delay>
    <delay out="deviceA">randomWalk(1ms,normal(0ms,0.5ms),"downlinkA")</delay>
</delays>
```

Listing 4: Example configuration of the PairwiseDelayer using the randomWalk function.

In Section 5.1.2, we present simulation results using our random walk PD.

3 Simulation Models for the Network Control Plane

The focus of the first release is on the data plane simulation rather than the control plane. Therefore, we only briefly present here, how to statically configure the data plane elements and a simple mechanism to change parameters at simulation time.

The existing TSN mechanisms implemented by INET, such as the gate control lists of the TAS, can be statically configured through configuration (ini) files. Listing 5 shows an excerpt from a configuration file shows an example configuration of a TAS with two gates of a TSN bridge. Figure 13 shows the resulting schedule.

```
*.switch.eth[*].macLayer.queue.numTrafficClasses = 2

# best-effort
*.switch.eth[*].macLayer.queue.transmissionGate[0].offset = 0ms
*.switch.eth[*].macLayer.queue.transmissionGate[0].durations = [4ms,6ms] #10ms period

# time-critical traffic
*.switch.eth[*].macLayer.queue.transmissionGate[1].offset = 6ms
*.switch.eth[*].macLayer.queue.transmissionGate[1].durations = [2ms,8ms] #10ms period
```

Listing 5: Example TAS configuration for a TSN bridge with two gates.

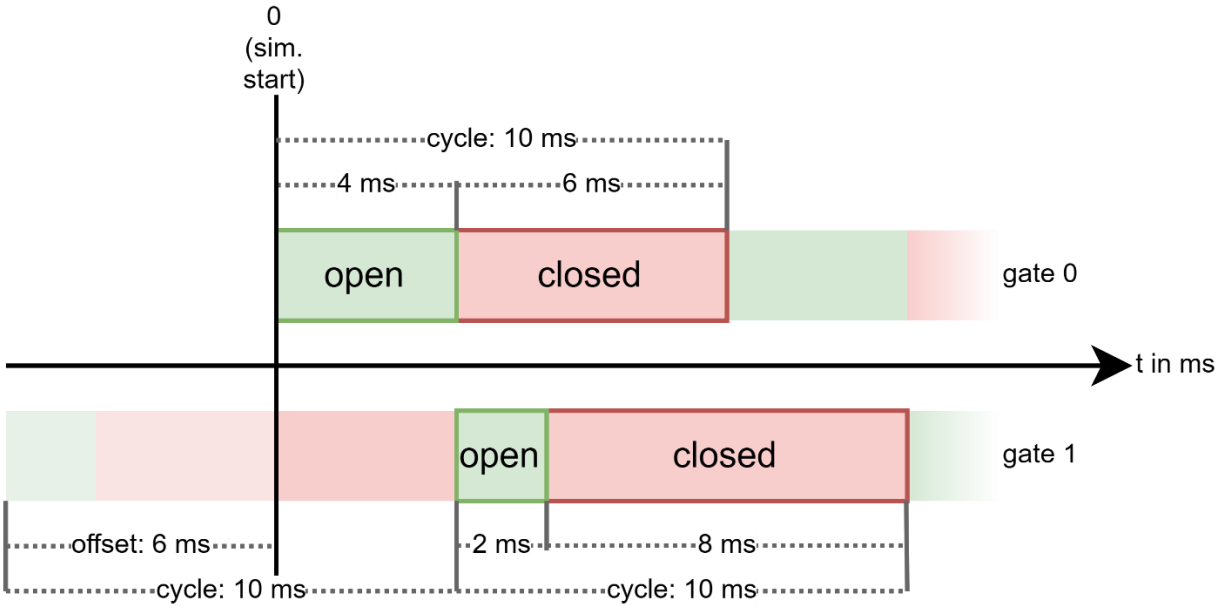

Figure 13: Resulting schedule from the configuration in Listing 5.

For the currently designed algorithms for IEEE 802.1Qbv schedule calculation, which takes a static set of streams as input and calculates an IEEE 820.1Qbv schedule that respects a specified PD (cf. deliverable D3.1 [DET23-31]), this static configuration is sufficient since these algorithms calculate the schedule for a set of streams and static parameters known a priori. An interface between 6GDetCom node (wireless TSN bridge) and Centralized Network Controller (CNC) will be added later, when scheduling algorithms are considered that can adapt to dynamic changes of PDs.

As already presented in Section 2.2.2, the PD and processing delay distributions can be configured through configuration files as well, which are read at the start of the simulation. This allows for static definitions of delay distributions.

Moreover, delay distributions or distribution parameters can be changed during simulation time at specific points in time through the so-called Scenario Manager[8] of INET. For instance, this allows for simulating a change of wireless link quality with respect to delay, as could be caused by a growing number of packet retransmissions with Hybrid Automatic Repeat Requests (HARQ protocols).

[8] Scenario Scripting — INET 4.5.0 documentation (omnetpp.org)

Besides this so far simple control plane configuration, the latter release of the simulation framework will also incorporate detailed control plane functions like a CNC.

4 Simulation Models for the Edge Cloud

Since we consider dependable end-to-end communication between applications – possibly containing a full round-trip from the initial talker to a listener (request) and back (response) with processing at the listener side, we also need to consider the infrastructure hosting these applications with respect to their timing behavior. Edge computing infrastructures are a prominent environment for time-sensitive applications to host applications physically close to end systems to avoid the inevitably long(er) propagation delay to a remote cloud infrastructure.

In this section, we present our approach to simulate the processing delay of applications hosted in an edge cloud infrastructure and exemplary data sets.

4.1 Alternative Approaches and Design Rationale

Similar to the approach to simulate PD, we can again identify two different approaches to simulated processing delay:

1. **Fine-grained simulation model** of all components of the execution environment contributing to processing delay. In modern edge cloud systems, the execution environment can be complex, including not only the physical machine, but also virtual machines controlled by a hypervisor, operating system (OS) kernel implementing different scheduling policies (e.g., strict priority scheduling, FIFO, round-robin, EDF, completely fair scheduling for Linux), containers executing in namespaces within an OS kernel.
2. **Coarse-grained simulation model picking processing delays from a given stochastic processing delay distribution**.

The advantages and disadvantages of both approaches are basically the same as for the simulation of PDs. Obviously, the complexity of simulating the processing delay with a fine-grained model is high. In addition, the existing INET framework is tailored to network simulation, and does provide only rudimentary support for the simulation of applications, let alone the whole execution environment.

Moreover, implementing the second approach is consistent with the simulation of PDs allowing for the re-use of some parts of the implementation. Therefore, we decided to implement the second approach enabling the specification of stochastic processing delay distributions in the simulator, which randomly delays the response to requests received by applications.

4.2 Design and Implementation of Processing Delayer

To add processing delay in edge cloud applications, we implemented a new application component for the simulator. Applications (“Apps”) in INET act as traffic sources and sinks. Since we focus on connection-less communication as it is typical for most real-time communication, the INET UDP application models are a suitable choice to base our application on. Therefore, we a new application model called *UdpEdgeCloudApp*[9], which is based on the *UdpEchoApp* of INET and can be used anywhere where other UDP apps (e.g., *UdpApp*, *UdpBasicApp*) from the INET framework can be used. The *UdpEdgeCloudApp* can be used as a subcomponent in different network devices that host traffic sources and sinks. The *UdpEdgeCloudApp* receives a UDP packet, delays it, and then forwards it to

[9] UdpEdgeCloudApp (deterministic6g.github.io)

another application, which can either be the initial sender or any other device, whose destination IP address and port number can be specified in the configuration as shown in Listing 6.

Figure 14 shows an exemplary network with a device called *cloudServer*, which implements an *UdpEdgeCloudApp*[9]. The delay is added within the C++ code of the *UdpEdgeCloudApp* before forwarding the packet to the configured device. Note that several apps can be hosted on the same device, therefore, *app[0]* as shown in Figure 14 (orange circle) specifies on of these apps.

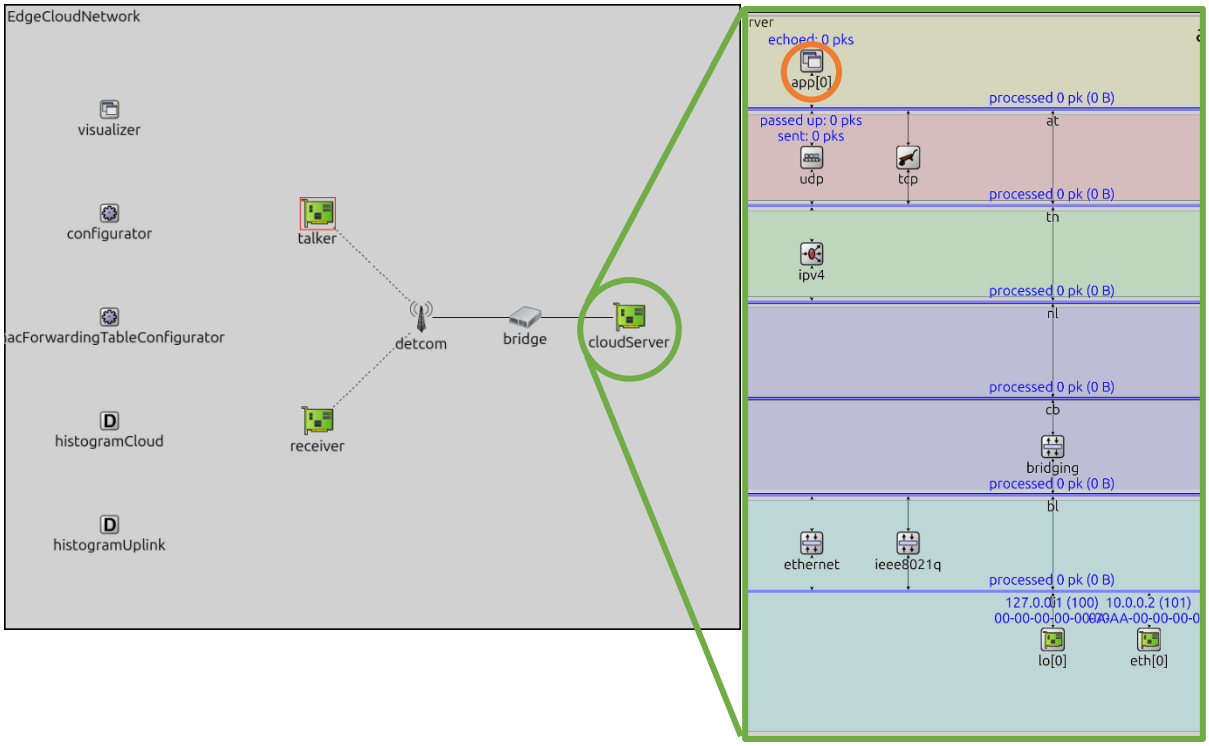

Figure 14: Example network with a cloud device using the UdpEdgeCloudApp (app[0]).

The processing delay configuration works similar to the configuration of PDs as described in Section 2.2.2, and can either be a custom NED-function, an OMNeT++ built-in statistical distribution, a mathematical expression, or an arbitrary combination of these.

Listing 6 shows the configuration of the *cloudServer* device in Figure 14, where *localPort* is the port on which the *UdpEdgeCloudApp*[9] is listening, and *destAddress* and *destPort* are the name of the destination device and destination port of the destination device, respectively. The *delay* parameter contains the intended delay configuration. In particular, every app of the cloud server has exactly one *delay* configuration, which is applied if a packet arrives at the specified *localPort*. In order to specify different delays for different sources, one can create multiple apps with a different *localPort* and modify the source apps respectively.

```
*.cloudServer.numApps = 1
*.cloudServer.app[0].typename = "UdpEdgeCloudApp"
*.cloudServer.app[0].destAddress = "receiver"
*.cloudServer.app[0].destPort = 1002
*.cloudServer.app[0].localPort = 1001
*.cloudServer.app[0].delay = rngProvider("histogramCloud")
```

Listing 6: Example configuration of a device using the UdpEdgeCloudApp.

If we execute a simulation using Figure 15a as an input distribution *histogramCloud*, we can see the resulting end-to-end delay of this stream follows the same distribution (Figure 15b), i.e., the delay is added as intended. Note, that the only network delay simulated in this simulation is the constant delay of Ethernet links, i.e. there is no PDV simulated by our *PairwiseDelayer*.

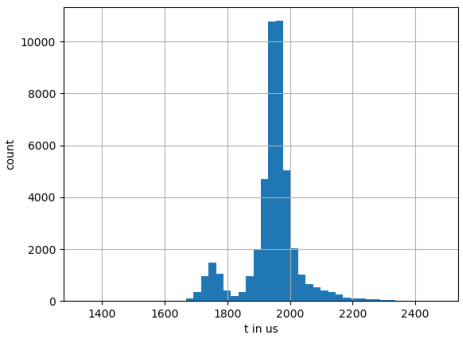

Figure 15a: Input distribution for the edge cloud delay.

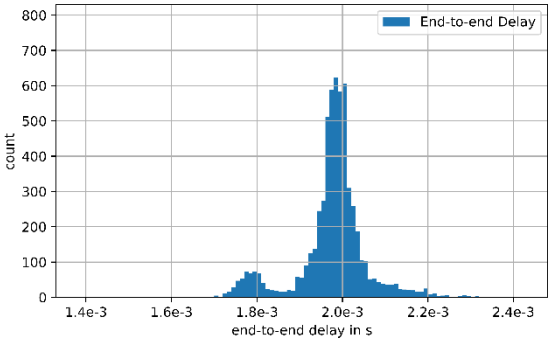

Figure 15b: Resulting end-to-end delay.

Figure 15: Comparison of the input PD for an edge cloud application and the resulting end-to-end delay.

4.3 Exemplary Processing Delay Distributions for the Edge Cloud

Together with the software to simulate processing delay distributions of edge cloud applications and services, we provide two data sets for modeling of processing delay (links to the datasets are provided in Table 1):

- ProcessingDelayDistribution1 (data set): This data set was measured in a smaller private edge cloud infrastructure.
- ProcessingDelayDistribution2 (data set): This data set was measured on a server node using a containerized application deployment.

4.3.1 ProcessingDelayDistribution1

Figure 16 shows the setup and how delay was measured. The environment for this edge cloud data set consists of one edge cloud server in a private edge cloud infrastructure. This edge cloud server is connected to one O-RAN indoor base station connected to a 5G core. The core is also connected to the public Internet where another public cloud server is operating.

Two mobile devices (smart phones) are communicating through the base station with the private edge cloud server and the public cloud server, respectively. The ping tool was used to measure the round-trip delays. All devices have synchronized clocks.

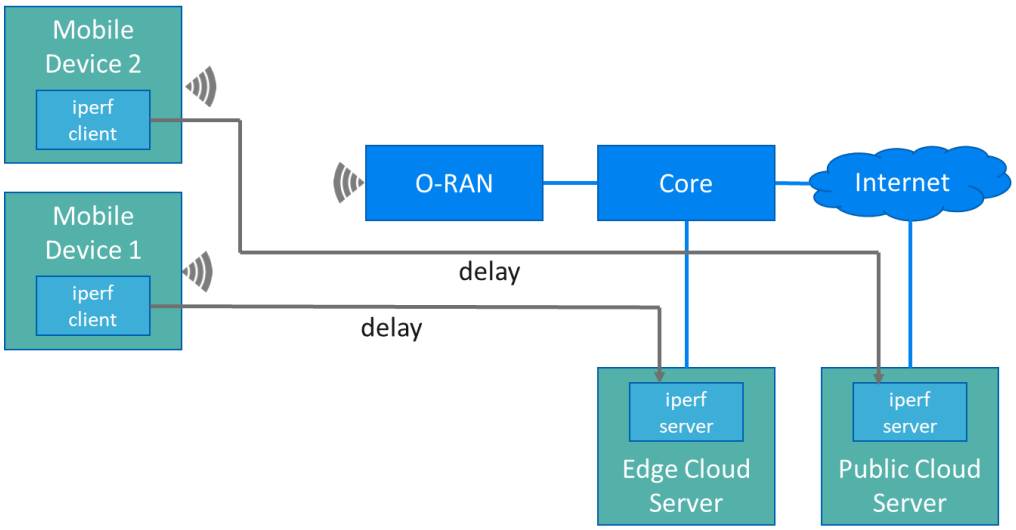

Figure 16: Measurement setup

4.3.2 ProcessingDelayDistribution2

The objective of this measurement experiment was to collect information about the response time of a cloudified application, considering the effect of the different delay components of the cloud execution environment, e.g., the application processing time, operating system, virtualization ecosystem, type of scheduling.

An application was used to simulate a collaborative, cloud-based robot control application. The application comprises three components, arranged as a chain of functional steps of the application; the first component could simulate video-processing/object tracking module, the second component represented the collaborative robot control logic, while the third component modeled the low-level, individual robot control.

For the cloud infrastructure, a server with multiple CPU and Real-time Linux OS was used. The application components were deployed in docker containers, and multiple instances of the first and third components were deployed. In the used experiment each component had deterministic execution time.

The performance of two types of scheduling, namely SCHED_FIFO and SCHED_DEADLINE was compared, and the resulting histograms is based on cumulative results of multiple measurements.

5 Simulation Scenarios

Various simulation scenarios are presented in our simulation framework to validate the effectiveness or evaluate the performance of the concepts and mechanisms designed in WP2 and WP3. In particular, the impact of packet and processing delay variations can be evaluated using the PD and processing delay simulation models presented in Section 2.2 and Section 4.2, respectively.

In the following subsections, we present the implementation of different simulation scenarios:

- A baseline scenario consisting of a simple line topology
- An industrial automation scenario based on a use case from WP1 (D1.1 *DETERMINISTIC6G Use Cases and Architecture Principles* [DET23-D11])
- Time synchronization scenarios tailored to the evaluation of gPTP

The first two scenarios focus on the performance of IEEE802.1Qbv scheduling with PDs.

The time synchronization scenario focuses on the performance of time synchronization with PDVs.

Please note that the presented results are preliminary and might change or will be refined until the final deliverable D4.5 *Validation Results*, which is explicitly focused on the presentation of validation results.

5.1 Baseline Scenario

The baseline scenario simulates a simple line topology with a single 6GDetCom node and further wired TSN bridges. Despite its simplicity, it can be used to showcase and test the core functions of the simulator framework, such as the previously described PairwiseDelayer component and the Histogram module.

5.1.1 Scenario Design and Implementation

In more detail, the Baseline scenario has the following topology (cf. Figure 17). On one edge of the network, we have a 6GDetCom node (*detcom*) with a wireless network and one wireless device (*device1*) as shown on the left side in the figure. The other side of the 6GDetCom node is connected to a line of wired TSN bridges with another end device (*device2*) at the end of the line topology.

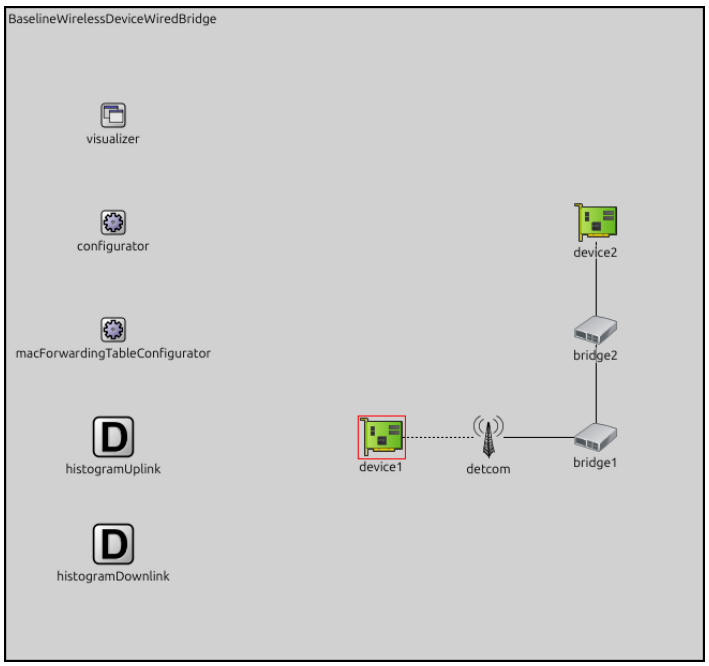

Figure 17: Baseline scenario with a wireless device and 6GDetCom node on the left side and a line of wired TSN bridges on the right side.

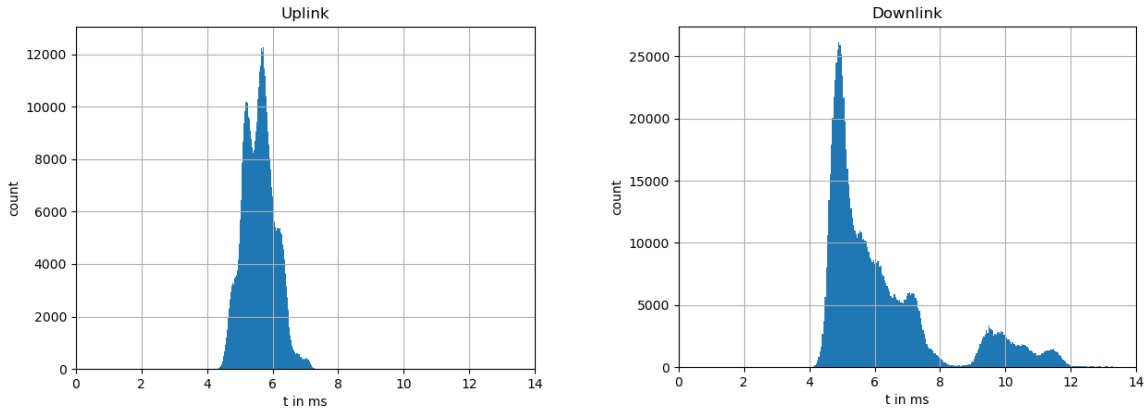

Figure 18: Given uplink distribution (left) and downlink distribution (right) for the Baseline scenario.

In this scenario, we use two different histograms shown in Figure 18 to configure PDs, one for the uplink direction of the wireless network and one for the downlink direction, respectively.

5.1.2 Results

We mainly use the Baseline scenario to explore the capabilities of our implemented components of the simulation framework, in particular, the *PairwiseDelayer*. This includes analyzing how the resulting simulated packet delay corresponds to the given input histograms of Figure 18. Figure 19 shows the distribution of the resulting packet delays in both, the uplink and downlink direction. By comparing the distributions from Figure 19 (result of the simulation) to the configured distributions in Figure 18, we see that the simulated distributions follow the given distributions.

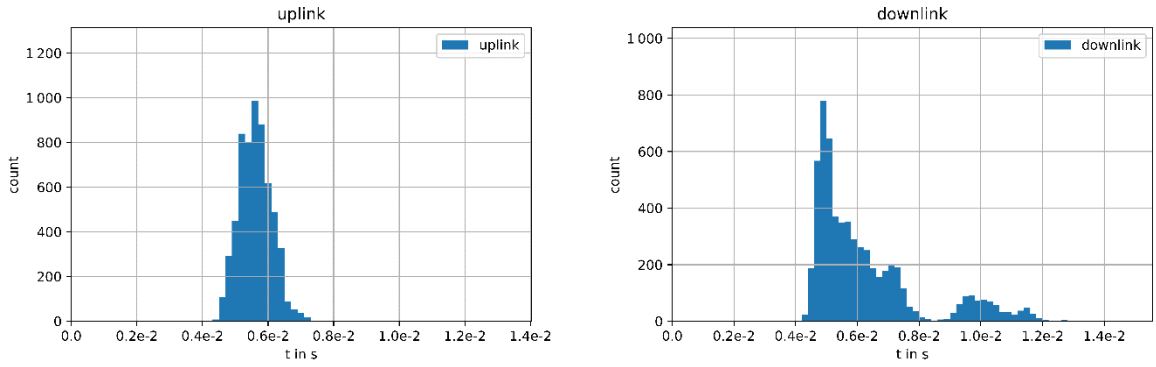

Figure 19: Resulting packet delays of the Baseline scenario.

Additionally, we use the Baseline scenario to showcase our random walk distribution as described in Section 2.2.3. As an exemplary configuration, we replace the downlink distribution with the random walk configuration shown in Listing 7.

```
randomWalk(5.5ms,normal(100us),"downlink")
```

Listing 7: Delay configuration using the randomWalk NED function.

Figure 20 shows the resulting end-to-end delay in uplink and downlink directions. The orange line depicting the downlink delay shows the characteristic random walk pattern centered around the mean value of 5.5 ms.

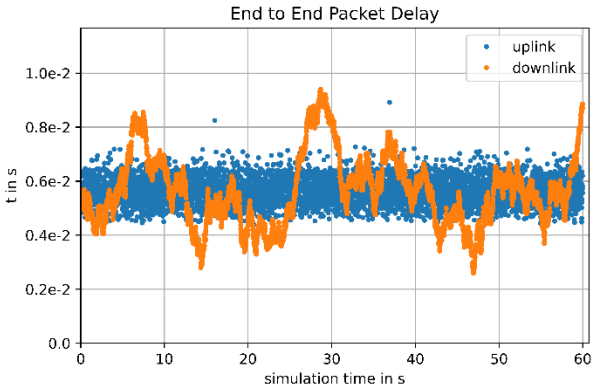

Figure 20: Resulting packet delays using random walk.

Further tests (not shown in this document) have been performed with other features to simulate PD as presented above to test their functionality. From these tests we conclude that the PD feature works correctly, i.e., the simulated PD follows the given PD distributions as configured.

5.2 Industrial Automation Scenario

The Industrial Automation Scenario has been derived from a use case of D1.1: *DETERMINISTIC6G Use Cases and Architecture Principles* [DET23-D11]. One purpose of this scenario is to validate and evaluate concepts for wireless-friendly end-to-end scheduling as designed in WP3. The design of such scheduling approaches is ongoing work at the time of writing this document. Therefore, we only present first preliminary evaluation results of one approach for wireless-friendly IEEE802.1Qbv scheduling with a TAS described in D3.1 *6G Convergence Enablers Towards Deterministic Communication Standards* [DET23-31], mainly to show the usefulness of the simulation framework to validate and evaluate such mechanisms.

This scenario consists of two Automated Guided Vehicles (AGVs), which connect wirelessly to a processing cell when they are close to the cell. The AGVs can communicate with each other wirelessly to coordinate their movement as a swarm, and they can communicate with the cell to coordinate the movement relative to the components of the cell. These movements could implement processing steps like bending, turning, inserting, or connecting individual or multiple parts. Although the AGVs are mobile in this scenario and the connection might be intermittent (only when the AGVs are close to the cell and registered with the cell), our first implementation of the scenario does not include the simulation of mobility. Therefore, we assume that the AGVs are constantly connected to the processing cell.

5.2.1 Scenario Design and Implementation

The topology of the scenario is depicted in Figure 21. It consists of one 6GDetCom node (*detCom*[10]), which connects wirelessly to two devices (*agv1* and *agv2*) that represent the AGVs. Moreover, the 6GDetComNode has a wired connection to a processing cell (*processingCell*). Note that each AGV and the processing cell contains a wired TSN network segment as shown in Figure 22, which illustrates the details of the AGV component (left figure) and the processing cell component (right figure) to show the sub-components implemented by the AGV and processing cell. The network segment for an AGV consists of a default INET TsnSwitch (*detComAdapter*) and two TsnDevices (*interAgv* and *interToCell*)

[10] As modules in Omnet++ need to start with a letter, we call it detCom instead of 6GDetCom node.

which send and receive packets. The processing cell consists of two TsnSwitches (*detComAdapter* and *bridge*) and two TsnDevices (*swarmControl* and *swarmStatus*), which also send and receive data.

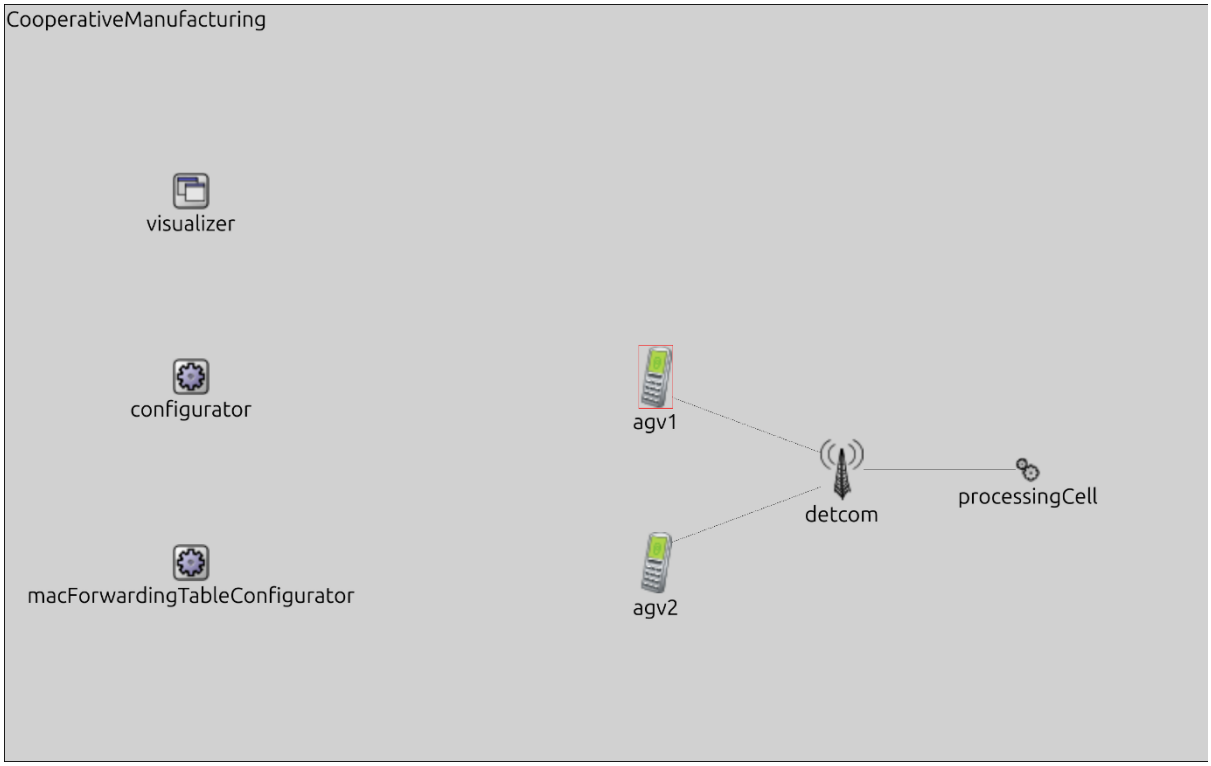

Figure 21: Industrial Automation Scenario – Topology

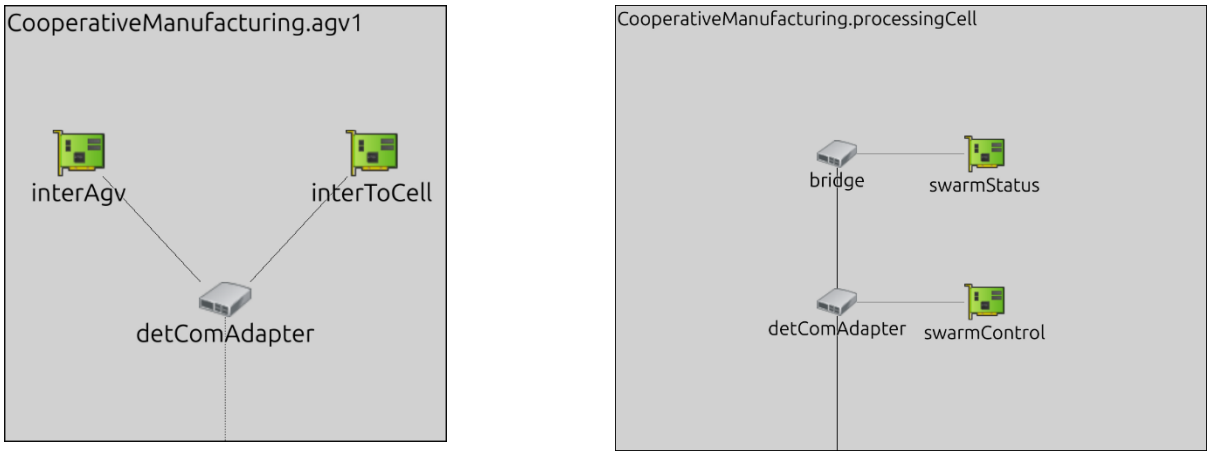

Figure 22: Details of the industrial automation scenario and its network segments of the AGVs and processing cell, respectively.

The traffic in this scenario includes six different streams:

1. Stream 1: From *agv1.interToCell* to *processingCell.swarmStatus*
2. Stream 2: From *agv2.interToCell* to *processingCell.swarmStatus*
3. From *agv1.interAgv* to *agv2.interAgv*

4. From *agv2.interAgv* to *agv1.interAgv*
5. From *processingCell.swarmControl* to *agv1.interToCell*
6. From *processingCell.swarmControl* to *agv2.interToCell*

This setup allows for analyzing the influence of PDVs onto streams and explore the capabilities of common and wireless-friendly scheduling algorithms as shown in the next sub-section.

5.2.2 Preliminary Results

In this sub-section, we discuss the first results from the Industrial Scenario evaluation. We first simulate scheduling with a TAS schedule calculated under the assumption of constant packet delay. Although this is an unrealistic assumption, it is still often made in existing scheduling approaches for wired TSN networks. Then, we show the impact of PDV onto the schedule calculated for ideal assumptions. Finally, we show how a wireless-friendly schedule will improve the robustness to PDV.

For all following scenarios, all streams have a cycle time of $1\ ms$ and all Ethernet links have a link speed of $100\ Mbps$ and a propagation delay of $50\ ns$. The size of all frames including all headers is $1000B$.

Influence of PDVs onto other streams

In our first evaluation, we only consider the first two streams, i.e., from *agv1.interToCell* to *processingCell.swarmStatus* and from *agv2.interToCell* to *processingCell.swarmStatus*. First, we assume constant transmission and propagation delay for both streams using the delay configuration for the 6GDetCom node (*detCom*) shown in Listing 8. Note, that these delay values include all delay components of the wireless links (dotted lines in Figure 21), i.e., also the propagation delay, transmission delay, delay because of retransmissions, etc. All other links (solid lines in Figure 21) simulate the delay of Ethernet links as described above.

```
<delays>
    <delay in="agv1">100us</delay>
    <delay in="agv2">100us</delay>
</delays>
```

Listing 8: Delay configuration using a constant delay for the *detCom* node.

We then use a “non-wireless-friendly” scheduling approach similar to [DN16], designed for wired TSN networks, assuming very small and constant delay, i.e. zero PDV. This approach aims to minimize the end-to-end delay and keeps the streams close together in time (“back-to-back” scheduling) to minimize the required number of GCL entries. With this scheduling approach, the calculated schedule is show in Table 2.

stream	sender	receiver	start	$\text{end}_{\text{transmission}}$	$\text{end}_{\text{propagation}}$
1	agv1. interToCell	agv1. detComAdapter	$0\ \mu s$	$80\ \mu s$	$80.05\ \mu s$
	agv1. detComAdapter	**detCom**	$80.05\ \mu s$	$80.05\ \mu s$	$\mathbf{80.05\ \mu s}$
	detCom	processingCell. detComAdapter	$\mathbf{180.05\ \mu s}$	$260.05\ \mu s$	$260.10\ \mu s$
	processingCell. detComAdapter	processingCell. bridge	$260.10\ \mu s$	$340.10\ \mu s$	$340.15\ \mu s$
	processingCell. bridge	processingCell. swarmStatus	$340.15\ \mu s$	$420.15\ \mu s$	$420.20\ \mu s$

2	agv2. interToProcessingCell	agv2. detComAdapter	80.96 μs	160.96 μs	161.01 μs
	agv2.detComAdapter	**detCom**	161.01 μs	161.01 μs	**161.01 μs**
	detCom	processingCell. detComAdapter	**261.01 μs**	341.01 μs	341.06 μs
	processingCell. detComAdapter	processingCell. bridge	341.06 μs	421.06 μs	421.11 μs
	processingCell. bridge	processingCell. swarmStatus	421.11 μs	501.11 μs	501.16 μs

Table 2: Schedule calculated by scheduler for wired networks.

Note, that Stream 2 starts later than Stream 1 to ensure there is no overlap on the first consecutive link from *detCom* to *processingCell.detComAdapter*. Stream 2 is scheduled directly after the transmission of Stream 1 except for the inter-frame gap (IFG) of $0.96\ \mu s$ in our setup. Note, that there is no transmission and propagation delay on the wireless link before the *detCom* node. These delays are part of the delay configured in Listing 8 and are shown as bold entries in Table 2. The calculated schedule leads to the GCL configuration for the *detCom* node as shown in Listing 9.

```
initiallyOpen = true
offset = 819.95us
durations = [161us,839us]
```

Listing 9: GCL configuration based on a non-wireless-friendly scheduling approach.

Using the *offset* parameter, we can set at which point in time of the cycle we are at the start of the simulation. As the first frame arrives at the gate at $180.05\ \mu s$, we want to open the gate at exactly this time. This means the offset needs to be $180.05\ \mu s$ before the start of any cycle. With our cycle time of $1000\ \mu s$ this leads to an offset of $1000\ \mu s - 180.05\ \mu s = 819.95\ \mu s$.

The simulation results in Figure 23a show that all packets arrive at their pre-calculated time with the expected end-to-end delay of $420.2\ \mu s$. Note, as the end-to-end delay is equal for both streams only one stream is visible in the diagram.

Obviously, the assumption of constant delays is unrealistic already in wired TSN networks but even to a larger extent in wireless networks. Therefore, next we add PDV to observe the impact on scheduling. We add PDV to Stream 1 and observe its impact on Stream 2 without additional PDV, using the previous schedule calculated for constant delays. To this end, we re-run the simulation with the delay configuration in Listing 10.

```
<delays>
    <delay in="agv1">normal(100us,10us)</delay>
    <delay in="agv2">100us</delay>
</delays>
```

Listing 10: Delay configuration with a PD distribution for one stream.

Without any changes to our schedule or the GCL this leads to the simulation results in Figure 23b. The figure shows, that the streams only arrive within their calculated time in the first cycle. In the second cycle, the frame of Stream 1 arrives at the GCL later than calculated leading to an end-to-end delay of $\sim 429\ \mu s$ (instead of the calculated $420\ \mu s$). Thus, by the time the frame of Stream 1 is completely

transmitted the remaining duration of the open gate is not long enough anymore to transmit the frame of Stream 2. This leads to the frame of Stream 2 being queued until the beginning of the next cycle. In the rest of the simulation, as soon as the gate opens in the next cycle, the queued frame of Stream 2 of the previous cycle is transmitted first (resulting in and end-to-end delay of $\sim 1340\ \mu s$). During this transmission, the frame of Stream 1 of the current cycle arrives which is then transmitted immediately afterwards. As the open gate slot is completely filled with the frame of Stream 2 from the previous cycle and the frame of Stream 1 of the current cycle, the frame of Stream 2 of the current cycle again has to be queued until the gate opening in the next cycle. Without dropping any packets (e.g. by using ingress filtering and policing) or using additional gate opening times, there is no possibility to recover from this.

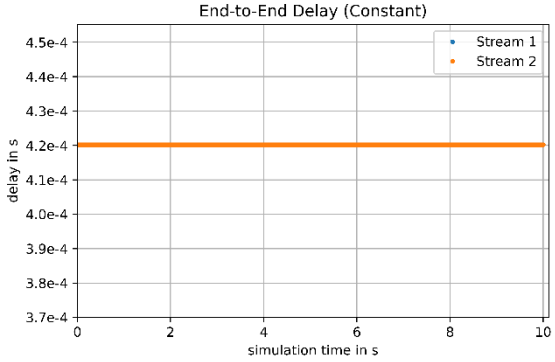

Figure 23a: Constant delay.

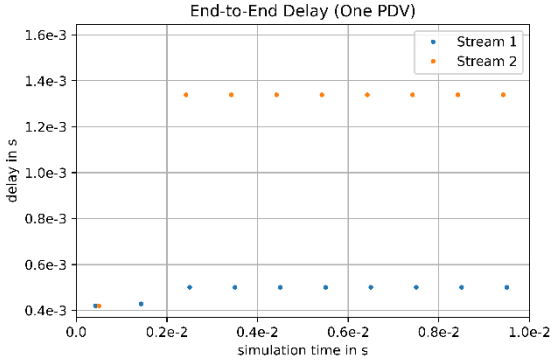

Figure 23b: Stream 1 with PDV.

Figure 23: Influence of PDVs onto other streams

Adapting the GCL on the DetCom Node by Considering PDV

In a third simulation, we analyze what happens if we adapt the GCL according to the PDV of Stream 1. In this run, we want to guarantee that Stream 1 arrives within the same cycle with a reliability of 99.7 %. For our chosen normal distribution with $\mu = 100\ \mu s$ and $\sigma = 10\ \mu s$ this corresponds to a PDV interval of $[\mu - 3\sigma, \mu + 3\sigma] = [70\ \mu s, 130\ \mu s]$. However, with a small probability, packets could still arrive outside this interval. One approach to protect other streams is to drop frames arriving outside of this interval. To this end, we could use the filtering feature from IEEE 802.1Qci *Per-Stream Filtering and Policing* [IEEE17-8021Qci] implemented by INET. To mimic the behavior of filtering as done by IEEE 802.1Qci, we can simply truncate the PD distribution using the configuration from Listing 11. Then no packets are sent that have delays outside the filtering time window.

```
<delays>
    <delay in="agv1">min(130us,max(70us,normal(100us,10us)))</delay>
    <delay in="agv2">100us</delay>
</delays>
```

Listing 11: Delay configuration with a truncated PD distribution for one stream.

We now need to reconfigure the GCL to open $30\ \mu s$ earlier and stay open for an additional $30\ \mu s$ as shown in Listing 12.

```
initiallyOpen = true
offset = 849.95us
durations = [221us,779us]
```

Listing 12: GCL configuration adapted to the configured PD distribution.

Figure 24 shows the simulation results of this simulation. We can see that the end-to-end delay of all streams stay within the cycle time. However, we can also see that the PDV of Stream 1 affects the arrival time of Stream 2. Without any additional knowledge about the required transmission guarantees of Stream 2 this might be problematic. Even if the receiver of Stream 2 could handle this additional delay, Stream 2 might now affect the delay of other streams on other links leading to a cascading effect.

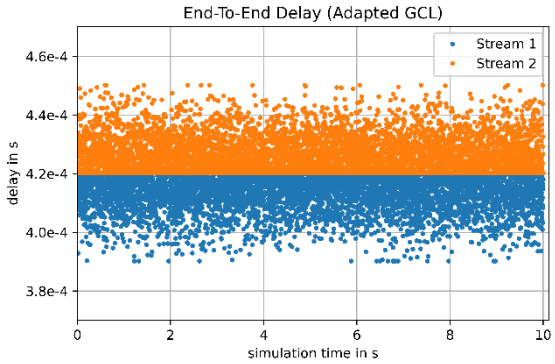

Figure 24a: E2E delays of Stream 1 and 2.

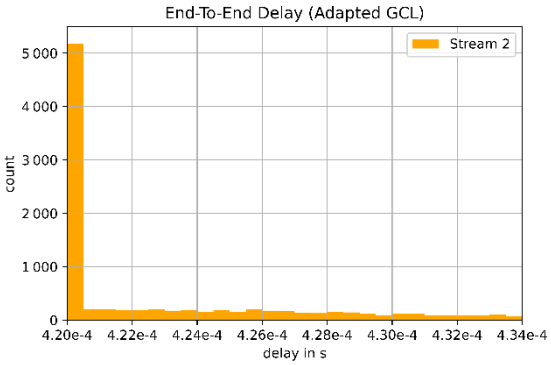

Figure 24b: Distribution of E2E delay of Stream 2.

Figure 24: Influence of an adapted GCL on streams with a PDV.

Using a Wireless-friendly Schedule

In a next simulation, we aim to adapt the schedule to mitigate the effects of the PDV of Stream 1. To this end, we use a robust, wireless-friendly scheduling algorithm as described in D3.1 *6G Convergence Enablers Towards Deterministic Communication Standards* [DET23-D31]. The objective of this algorithm is to maximize the gap between all streams while maintaining a low end-to-end delay. In our scenario, this leads to the "wireless-friendly" schedule shown in Table 3.

stream	sender	receiver	start	$end_{transmission}$	$end_{propagation}$
1	agv1. interToCell	agv1. detComAdapter	0 μs	80 μs	80.05 μs
	agv1. detComAdapter	**detCom**	80.05 μs	80.05 μs	**80.05 μs**
	detCom	processingCell. detComAdapter	**180.05 μs**	260.05 μs	260.10 μs
	processingCell. detComAdapter	processingCell. bridge	260.10 μs	340.10 μs	340.15 μs
	processingCell. bridge	processingCell. swarmStatus	340.15 μs	420.15 μs	420.20 μs
2	agv2. interToCell	agv2. detComAdapter	500 μs	580 μs	580.05 μs
	agv2. detComAdapter	**detCom**	580.05 μs	580.05 μs	**580.05 μs**
	detCom	processingCell. detComAdapter	**680.05 μs**	760.05 μs	760.10 μs
	processingCell. detComAdapter	processingCell.bridge	760.10 μs	840.10 μs	840.15 μs

	processingCell.bridge	processingCell.swarmStatus	840.15 μs	920.15 μs	920.20 μs

Table 3: Wireless-friendly schedule

Please note that Stream 2 is now scheduled exactly $500\ \mu s$ after Stream 1, which is exactly half the cycle time of $1\ ms$. For two streams, this is the optimal case, as it minimizes the probability of streams colliding with streams of the same cycle as well as the next cycle. Please note that in this example, an adaptation of the start times at the end systems is sufficient to avoid interference between streams. However, in general, a wireless-friendly schedule will consider both, the start times of transmissions at end systems as well as the transmission times at bridges as defined by the GCL. The adapted GCL for this schedule looks like shown in Listing 13.

```
initiallyOpen = true
offset = 849.95us
durations = [140us,390us,80us,390us]
```

Listing 13: GCL configuration based on a wireless-friendly scheduling approach.

In Figure 25, we can see that the PDV of Stream 1 now does not have an influence on the end-to-end delay of Stream 2 anymore.

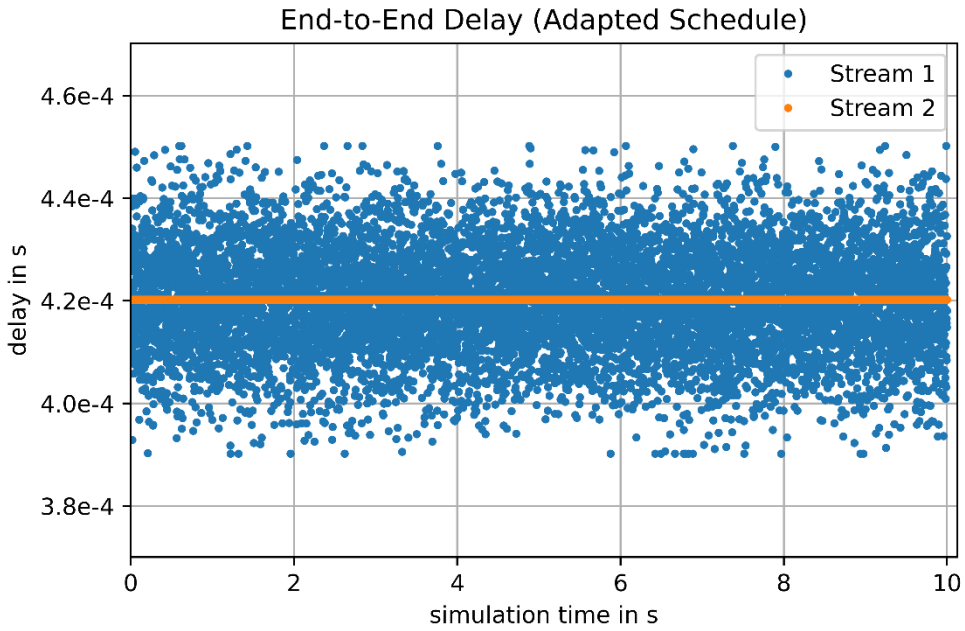

Figure 25: End-to-end delay of two streams in a scenario with an adapted scheduling algorithm.

In conclusion, this shows:

- With the simulation framework, we are able to investigate the impact of PDVs on scheduling. In particular, we can analyze the impact of PDV onto scheduling as shown and the improvements made by novel wireless-friendly scheduling algorithms. However, evaluations are not limited to analyzing scheduling under PDV, but can also be used for analyzing other time-dependent mechanisms such as time synchronization as shown in the next sub-section.
- As a first preliminary evaluation result, we see that the robustness of time-driven IEEE 802.1Qbv schedules can be improved through wireless-friendly schedules. More concepts for robust,

wireless-friendly scheduling will be investigated in the future and validated by further enhancing this and other scenarios.

5.3 Time Synchronization Scenario

The time synchronization scenario of our simulation framework is used to analyze how gPTP performs in converged 5G/6G-TSN networks, in particular, when adding PDV. For TSN networks, INET already supports time synchronization using the gPTP standard including showcases[11] for different kinds of time synchronization networks. In this section, we describe how we can adapt and use the gPTP functionality of INETs in our simulation framework.

As described in D2.2 *Time synchronization for E2E time awareness* [DET23-D22] and also shown in Figure 26, converged 5G-TSN networks use two different time synchronization domains: the 5G time synchronization domain and the TSN time synchronization domain, respectively. The 5G network components (UE, UPF) are synchronized within the 5G time synchronization domain. The TSN time synchronization domain is synchronized to a time transmitter using the gPTP protocol.

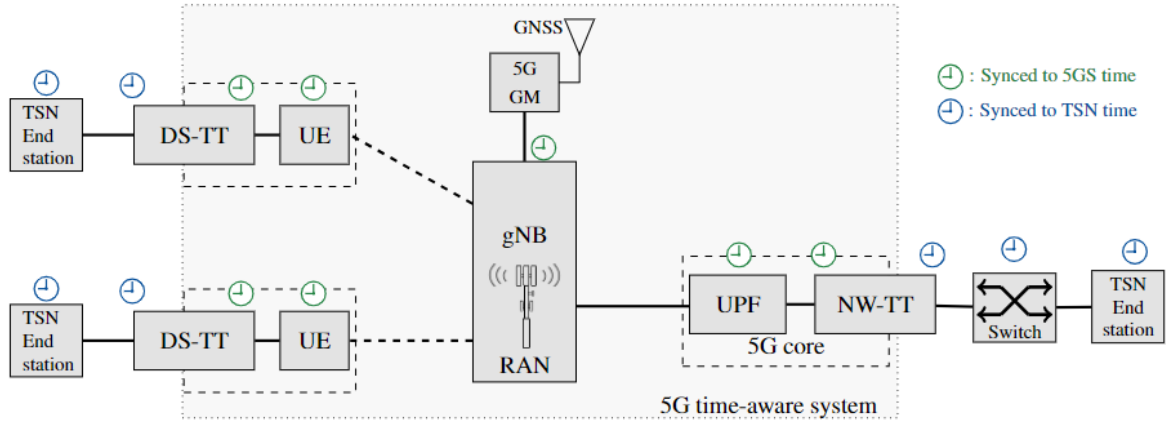

Figure 26: Time synchronization in 5G-TSN networks

As our 6GDetCom node is based on the INET TsnSwitch, we can use its built in gPTP component in *Bridge* mode. Figure 27 shows this setup using a time transmitter and two TSN devices synchronizing to the time transmitter (time receivers). The depicted diff values show the time difference between time transmitter and time receivers.

[11] Using gPTP — INET 4.5.0 documentation (omnetpp.org)

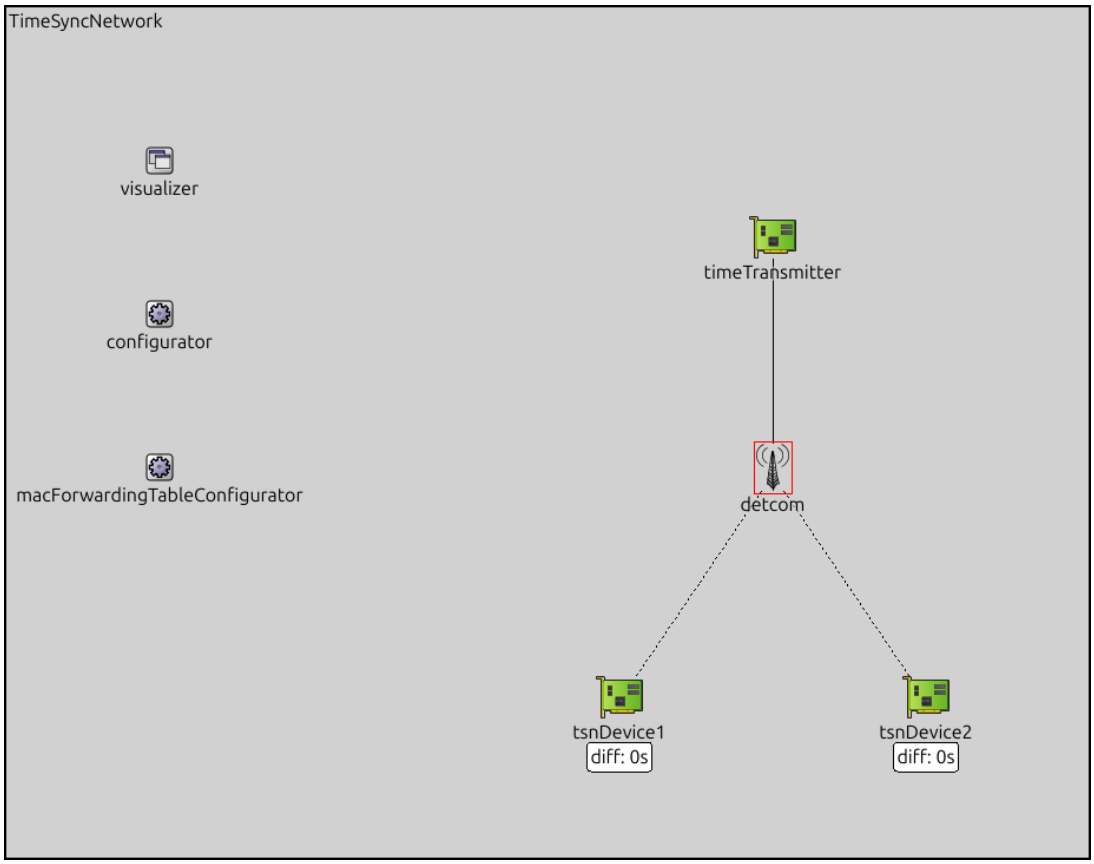

Figure 27: TimeSynchronization showcase using our 6GDetCom node.

As described in more detail in D2.2 [DET23-D22], the 6GDetCom node has to be aware of gPTP frames and calculate the residence time of gPTP frames within the 6GDetCom node. This works by setting an ingress timestamp t_i, when a gPTP frame enters the 6GDetCom node and an egress timestamp t_e, when the gPTP frame leaves the 6GDetCom node. The residence time within the 6GDetCom node can then be calculated as $t_e - t_i$, which is then added to the *correctionField* of the gPTP frame. This is consistent with the definition of TSN bridges and, thus, should work out-of-the-box with the INET framework and in our network presented in Figure 27.

However, we discovered two problems with the current INET implementation of gPTP and the current simulation model. First, the INET framework currently has an issue with its gPTP implementation which does not set *correctionField* correctly. This problem is already under investigation by the developers of INET[12] but has not been fixed yet. Secondly, the abstraction made above assumes perfectly synchronized clocks within the 6G domain, i.e., t_i and t_e are synchronized within the 6G domain. However, as D2.2 describes, the clocks in a 6G domain are not perfectly synchronized – note that in general there is a wireless link with stochastic delay between the components taking the timestamps t_i and t_e. This means that t_i and t_e are set by different (usually not perfectly synchronized) clocks which leads to an inaccurate calculation $t_e - t_i$ of the residence time.

Due to these two problems, we refrain from presenting preliminary results here, which would not be consistent with a real implementation. Instead, we will tackle both problems in future work to gain

[12] 802.1as · Issue #920 · inet-framework/inet · GitHub

meaningful simulation results for the time synchronization scenario. First of all, we will fix the INET gPTP implementation to correctly set *correctionField*. Secondly, we will add a realistic clock model for the 5G domain in order to simulate the inaccurate calculation of the residence time, which impacts the accuracy of clock synchronization. These extensions are subject to a future release of the simulation framework.

6 Conclusions & Future Work

6.1 Conclusion

In this document, we provided an overview of the first open-source software release of the simulator framework used in the DETERMINSTIC6G project for validating concepts developed as part of the project. The main features of the first release are the simulation models to simulate stochastic PD in the data plane of TSN networks and processing delay of applications hosted in an edge cloud infrastructure. We presented the generic design of a Packet Delayer component, which can simulate PDV based on closed-form descriptions (probability distributions), data sets of measured PD data – in particular data measured as part of the project in a 5G testbed, and algorithmically with implementations of stochastic processes, enabling the simulation of correlated packet delay. A similar feature has been presented for simulating processing delay.

Moreover, we introduced a set of PD models and processing delay models, which are published together with the source code of the simulator.

Finally, we presented three validation scenarios: a baseline and an industrial scenario, which focus on validating the effects of PDV onto scheduling in TSN networks; and a third scenario, which focuses on evaluating the performance of time synchronization under PDV. As a first result, we showed how that wireless-friendly schedules have the potential to improved dependability under PDV.

6.2 Future Work

The first software release of the validation framework is only an intermediate step towards the final validation framework. A second release is planned, which includes the following items as part of future work:

Extended simulation models for the network data plane: The major feature of the network data plane implemented so far is the ability to simulate characteristic stochastic PD distributions. So far, we have only implemented independent and identically distributed (i.i.d.) random delay variables. However, delay might also be correlated, i.e., delay values which are close in time to each other might be similar. The next release will also allow for the simulation of non-i.i.d. PD distributions.

Moreover, so far, we only considered existing standard TSN mechanisms for scheduling, in particular, the TAS (IEEE 802.1Qbv). In the future, we will consider other TSN shapers (as already provided by INET), and also other novel scheduling mechanisms and data plane enhancements like Packet Delay Correction (PDC), which need to be implemented for the simulator.

Simulation models for the network control plane: So far, the network control plane models only allow for static configurations of the data plane mechanisms, mainly through configuration files. However, the adaptation to dynamic network conditions at runtime such as changing PD distributions is essential for wireless systems. For instance, a packet schedule might need to be adapted when delay distributions change significantly. To validate adaptation mechanisms, several extensions of the

validation framework are required: (1) a network control component (CNC in TSN parlance) has to be added to the simulator. Open-source CNC implementations already exist for real (physical) TSN networks, such as the OpenCNC [OPN23]. We will explore whether it is possible to create an interface for the simulator to connect such existing CNCs to the simulator via standard protocols like NETCONF [IETF11-RFC6241] for the so-called southbound interface between CNC and (now simulated) network elements (6GDetCom node). The major advantage of such an approach is to use the actual control logic as implemented for real networks in the CNC, while only simulating the network data plane. This also helps to evaluate the computational overhead, for instance, to calculate new schedules for IEEE 802.1Qbv, which is a very costly operation with respect to processing. This computational overhead is hard to evaluate through "pure" simulations, but relatively simple to measure. (2) The control logic to be executed by the CNC needs to be implemented. For instance, we will implement different algorithms to calculate robust schedules for IEEE 802.1Qbv and possibly also other scheduling mechanisms and execute them within the CNC to automatically configure the 6GDetCom nodes and TSN bridges along the end-to-end path. We will also investigate how the adaptation can be triggered through an extended event-driven southbound interface between network elements and CNC and with the help of algorithms estimating and predicting the PD distributions based on observed delay date.

Security: For dependable communication, security mechanisms to detect and mitigate attacks are of great importance. Extensions of the simulator shall enable to evaluate these security mechanisms. To this end, the simulator needs to be extended to allow for integrating packet traces to simulate attacks and implement the corresponding mechanisms to detect attacks.

Extended scenarios: The scenarios will be further extended and refined, for instance, including more traffic models and refined scenarios for time synchronization, fixing the problems identified above, adding clock models, and to simulate and validate standby mechanisms increasing the resilience of time synchronization.

Finally, the simulation framework will be used to produce extensive simulation results for validating the novel concepts that are developed during the project.

References

[3GPP16-22261]	3GPP TS 22.261, "Service requirements for the 5G system," v19.4.0
[AAB+22]	J. Ansari, C. Andersson, P. de Bruin, J. Farkas, L. Grosjean, J. Sachs, J. Torsner, B. Varga, D. Harutyunyan, N. König, R. H. Schmitt, "Performance of 5G Trials for Industrial Automation. Electronics", 2022; 11(3):412. https://doi.org/10.3390/electronics11030412
[AVK+22]	J. Ansari, B. Varga, P. Kehl, N. König, R. H. Schmitt "5G and TSN integrated prototype for flexible production", presentation at TSN/A conference, Stuttgart, September 28-29, 2022
[DET23-D11]	D1.1: DETERMINISTIC6G Use Cases and Architecture Principles
[DET23-D21]	D2.1: 6G Centric Enablers
[DET23-D22]	D2.2: Time Synchronization for E2E Time Awareness
[DET23-D31]	D3.1: 6G Convergence Enablers Towards Deterministic Communication Standards

[DN16]	F. Dürr, N. G. Nayak, No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN), In: 24th International Conference on Real-Time Networks and Systems (RTNS 2016), October, 2016
[IEEE09-8021Qav]	802.1Qav-2009 – IEEE Standard for Local and metropolitan area networks-- Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams, DOI: 10.1109/IEEESTD.2010.8684664
[IEEE11-8021AS]	802.1AS-2011 – IEEE Standard for Local and Metropolitan Area Networks – Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks
[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
[IEEE16-8021Qbu]	802.1Qbu-2016 – IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption, DOI: 10.1109/IEEESTD.2016.7553415
[IEEE17-8021CB]	802.1CB-2017 – IEEE Standard for Local and metropolitan area networks – Frame Replication and Elimination for Reliability, DOI: 10.1109/IEEESTD.2017.8091139
[IEEE17-8021Qci]	802.1Qci-2017 – IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 28: Per-Stream Filtering and Policing, DOI: 10.1109/IEEESTD.2017.8064221
[IEEE20-8021Qcr]	802.1Qcr-2020 – IEEE Standard for Local and Metropolitan Area Networks – Bridges and Bridged Networks – Amendment 34: Asynchronous Traffic Shaping, DOI: 10.1109/IEEESTD.2020.9253013
[IETF11-RFC6241]	RFC 6241 – Network Configuration Protocol (NETCONF), June 2011
[INE23]	INET Framework (website), https://inet.omnetpp.org/, last accessed December 31, 2023
[KAJ+22]	P. Kehl, J. Ansari, M. H. Jafari, P. Becker, J. Sachs, N. König, A. Göppert, R. H. Schmitt, “A Prototype of 5G Integrated with TSN for Edge-Controlled Mobile Robotics" Electronics 11, no. 11: 1666, 2022. https://doi.org/10.3390/electronics11111666
[MSG23]	S. Mostafavi, G. P. Sharma, J. Gross, Data-Driven Latency Probability Prediction for Wireless Networks: Focusing on Tail Probabilities, arXiv:2307.10648 , July 2023, DOI: 10.48550/arXiv.2307.10648
[NSS+20]	G. Nardini, D. Sabella, G. Stea, P. Thakkar, A. Virdis , Simu5G – An OMNeT++ library for end-to-end performance evaluation of 5G networks, IEEE Access, 2020, DOI: 10.1109/ACCESS.2020.3028550
[OMN23]	OMNeT++ Discrete Event Simulator (website), https://omnetpp.org/, last accessed December 31, 2023
[OPN23]	OpenCNC (Github repository), https://github.com/AIDA-KAU/OpenCNC, last accessed December 31, 2023
[S73]	H. Stachowiak, Allgemeine Modelltheorie, Springer Verlag, Wien – New York, 1973, ISBN 3-211-81106-0
[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

List of abbreviations

AGV	Automated Guided Vehicle
ARP	Address Resolution Protocol (RFC 826)
CNC	Centralized Network Controller
GCL	Gate Control List
gPTP	generalized Precision Time Protocol (IEEE 802.1AS)
HARQ	Hybrid Automatic Repeat Requests
i.i.d.	independent and identically distributed
NED	NEtwork Description
PD	Packet Delay
PDC	Packet Delay Correction
PDV	Packet Delay Variation
PTP	Precision Time Protocol (IEEE 1588)
RSTP	Rapid Spanning Tree Protocol (IEEE 802.1w)
TAS	Time-Aware Shaper (IEEE 802.1Qbv)
TSN	Time Sensitive Networking

Table 4: List of abbreviations

Terms and Definitions

6GDetCom node	A wireless TSN bridge including 6G mobile network components.
Delayer	A component of the simulated data path of an Ethernet bridge that adds delay to packets passing through the bridge.

Table 5: Terms and Definitions