Miranda Jr. *et al. J Wireless Com Network (2025) 2026:17*
https://doi.org/10.1186/s13638-025-02474-9

Journal on Wireless Communications and Networking

RESEARCH Open Access

A bottom-up approach for QoS-driven network slicing in TSN networks

Gilson Miranda Jr.[1,2*], Nina Slamnik-Kriještorac[2], Daniel Macedo[1] and Johann Marquez-Barja[2]

*Correspondence:
gilson.miranda@uantwerpen.be

[1] PPGCC, Federal University of Minas Gerais, Belo Horizonte, Minas Gerais, Brazil
[2] University of Antwerp-imec, IDLab-Faculty of Applied Engineering, Antwerp, Belgium

Abstract

Network slicing enables multiple virtual networks to share physical resources, allowing network operators to deliver highly customizable and efficient networking solutions that meet the diverse requirements of modern applications. Automating the deployment and runtime management of network slices is essential for supporting network scaling with minimal human intervention. With several applications with different performance requirements, the manual configuration of slices is infeasible. Different initiatives have investigated methods in which network operators define high-level Key Performance Indicators that a network must deliver to networked applications, while the details of *how* to achieve these goals are abstracted by underlying control systems. Many works in the literature present a top-down approach, focusing on the high-level decision processes and relying on abstracted infrastructure managers and simulation tools to apply/execute such decisions. In this work, we leverage components that we previously developed for network monitoring, traffic shaping, and Software-Defined Time-Sensitive Networking to create a bottom-up approach toward automated slice management driven by Quality of Service goals. We describe the intricate coordination of elements required for an automated control loop and present the results achieved with a proof of concept executed in a real testbed of wired and Wi-Fi nodes. The applications use the data models introduced in this work to inform their requirements to the control plane. The control plane correctly prepares the network by applying rules for monitoring, traffic classification, and traffic shaping, effectively creating slices with different performance profiles for each application. Lastly, we present results using our control loop that continuously monitors each flow, identifies issues affecting the network performance, and takes corrective actions to reestablish the intended performance level of the applications.

Keywords: QoS, TSN, Network slicing

1 Introduction

In industrial and multimedia use cases, different network applications like process control, video surveillance, alarm propagation, and object tracking often coexist, each with its own set of requirements. Network slicing (NS) enables the deployment of independent virtual end-to-end networks running over a shared physical infrastructure, relying extensively on virtualization mechanisms for resource sharing and isolation. The ability to create slices tailored to the performance requirements of different

applications makes NS a suitable solution to support the aforementioned industrial and multimedia use cases. However, there are several technical challenges to be addressed when managing network slices. The control plane is commonly unaware of the performance requirements and traffic characteristics of network applications, hindering appropriate allocation of resources and often leading to resource over-provisioning [1]. Moreover, the slice life cycle management and the assurance that they meet the performance requirements at all times must be automated to minimize the need for human intervention [2, 3]. Optimizing the resource allocation to each slice can improve the number of served slices over a single physical network and increase revenues [4].

With the evolution of Time-Sensitive Networking (TSN) over Ethernet, the convergence of Operational Technology (OT) and Information Technology (IT) networks is becoming more feasible [5]. In this context, the same network carrying regular traffic, e.g., from office users accessing websites, can be used to carry high-priority time-sensitive traffic such as alarms and machinery control. TSN originated from the multimedia domain with the Audio Video Bridging (AVB) Task Group by Institute of Electrical and Electronics Engineers (IEEE) 802.1 in 2007 and had its scope widened to support more applications [6]. The TSN standards have shown promising capabilities to effectively realize network resource allocation [7]. However, the standards addressing configuration are still in early development, with no well-established methods or systems for TSN control [8]. Still, TSN features enable a high level of control over network traffic traversing Network Elements (NEs) and can be leveraged to realize NS beyond industrial-only use cases.

Different initiatives from European Telecommunications Standards Institute (ETSI), 3rd Generation Partnership Project (3GPP), and 3rd Generation Partnership Project (3GPP), introduce data models, interfaces, and workflows to realize autonomous network configuration and management. Using the concept of Intent-Based Networking (IBN) [9], these initiatives refer to the concept of *intents* as:

- A set of operational goals (that a network should meet) and outcomes (that a network is supposed to deliver) defined in a declarative manner without specifying how to achieve or implement them [9]
- A desire to reach a certain state for a specific service or network management workflow. An intent specifies the expectations, including requirements, goals, and constraints, given to a 3GPP system, without specifying how to achieve them [10]

While NS and IBN have been the subject of several studies in the last years, many works focus on the high-level aspects, relying on abstracted infrastructure managers and simulation tools to evaluate the performance of the solutions [11–15]. Despite the contributions of such works, it is also important to address the tasks supporting these high-level operations to highlight the feasibility of the solutions over real devices. This also involves presenting *how* to deploy a solution that can deliver the performance levels requested by the applications.

In this work, we present a system for Quality of Service (QoS)-driven network slicing in a bottom-up perspective, building on top of our previous works addressing:

- Fine-grained network telemetry using In-band Network Telemetry (INT), enabling per-flow and per-hop monitoring by appending telemetry metadata to data packets [16].
- Software-Defined Networking (SDN) control for TSN networks, for flexible management of wired and Wi-Fi networks with TSN capabilities [7].
- Data plane programmability using softwarized TSN functions, extending the support for traffic shaping using virtualized elements, capable of operating consistently over wired and Wi-Fi devices [17].

With these previous works as enablers, we progressed to a higher-level management perspective, developing data models that allow applications to communicate their requirements and characteristics to the network, using insights from recent 3GPP and ETSI standards for Zero-touch network and Service Management (ZSM) and IBN. We adapted these models to integrate and operate with our TSN controller architecture and evaluated this system for managing slices over a mixed wired/Wi-Fi testbed, with an automated control loop that uses QoS measurements as feedback.

The paper is organized as follows: Sect. presents the related work. Section gives an overview of the system and the TSN configuration model used. Section describes the enablers that we introduced in previous works to support the automated deployment and runtime configuration of network slices. Section describes the data models that we developed to express the requirements and characteristics of network applications, giving a basis for automated decision-making processes. Section describes the control loop we implemented to continuously monitor network flows, identify issues, and take corrective actions. We report the performance evaluation results in Sect. . Section concludes this paper and discusses future works.

2 Related work

In this section, we first discuss the related works addressing network slicing mechanisms that aim to adapt the allocation of resources toward a user- or application-specified set of goals. In this context, works on IBN aim at managing network infrastructure to satisfy *intents*, usually expressed through a high-level language. Several works address the problem of expressing and interpreting intents and their translation into objective goals, metrics, or actions [11–15, 18].

For industrial or critical communications, Saha et al. [11] and Mehmood et al. [12] present architectures highlighting the interpretation of intents and extraction of key information from keywords. Lower-level actions are attributed to infrastructure managers such as OpenStack. Cerroni et al. [13] propose a reference architecture for the orchestration of heterogeneous SDN domains, with some details on JavaScript Object Notation (JSON) messages to express intent-based requests to a virtual infrastructure manager. These infrastructure managers abstract details about which techniques can enforce the decisions on the network, and the evaluation is done through emulation or simulation. In this work, we start from the low-level enablers for traffic monitoring and shaping on heterogeneous networks and then incrementally add the building blocks for high-level management. With this approach, we detail methods and techniques that can be used to build such systems, as well as the configuration sequences and workflows

necessary for it. Moreover, we perform an experimental evaluation of our system on a wired/Wi-Fi testbed.

Rothenberg et al. [14] and Perepu et al. [15] propose systems for intent-based multimedia management using Quality of Experience (QoE) feedback. Rothenberg et al. [14] target a video-streaming application, using a QoE feedback estimated using QoS models and QoS metrics. The authors introduce network impairments such as bandwidth fluctuation and packet reordering using Traffic Control (TC) scripts. The work by Perepu et al. [15] considers a Conversational Video application for which a QoE metric between [1, 5] must be maximized, and Ultra-Reliable Low-Latency Communication (URLLC) and mobile Internet of Things (IoT) applications for which the Packet Loss Rate (PLR) between [0, 1] must be minimized. The work focused on a Multi-Agent Reinforcement Learning system to control bitrate and priority for the flows and evaluated the performance using a network emulator. Our work uses objective QoS metrics as target Key Performance Indicators (KPIs) to be optimized, while QoE-based management is envisioned as a future step. Still, the QoS metrics used in our Proof of Concept (PoC) are obtained via in-band telemetry, which monitors the actual traffic flows between different nodes in our testbed.

Network slicing over Wi-Fi is also a relevant subject, as the collaboration between Wi-Fi and cellular (5 G/6 G) networks has been long envisioned [19]. Richart et al. [20] propose a slicing mechanism for Wi-Fi Access Points (APs) combining a queuing and scheduling mechanism. The solution is evaluated analytically and through simulations. Isolani et al. [3] carry out experimental work with a SDN-based approach for on-the-fly end-to-end slice orchestration. The solution is based on the framework from Coronado et al. [19], enhancing it with an algorithm that periodically adjusts the airtime allocation of the slices on a Wi-Fi AP. The system considers two slices: QoS and best-effort, being unclear whether more slices are supported. The resource allocation prioritizes the QoS slice; however, the authors do not address methods for the applications to request specific performance requirements to the network. In this work, we leverage the fully centralized network model for TSN networks and introduce data models that can be used by network operators or applications to request specific performance levels to the network in terms of throughput, delay, jitter, and PLR. Fami et al. [21] propose a Wi-Fi slicing system for IoT applications. The authors propose a centralized controller to manage the association and resource allocation for NEs. The authors aim to improve the aggregate throughput of the network and do not evaluate other metrics. The results were obtained via simulations and show a slight improvement in the optimized method over a system that controls node association based on Received Signal Strength Indicator (RSSI).

This work builds on our previous experimental works on SDN-based TSN networking and flexible data plane monitoring and configuration. We extend our previous solutions with data models based on 3GPP and ETSI specifications for IBN, providing the means for applications to inform their requirements to the network control plane. Moreover, we expanded the functionalities of our SDN controller and implemented a control loop module to continuously monitor and adjust the resource allocation of network slices. As we build on these previous works, our main focus is on industrial and multimedia applications; however, the solutions are also applicable to other domains such as office, campus, and other general-purpose networks.

3 System overview

The system described in this work is based on our TSN controller architecture introduced in our previous work [7]. Our TSN Controller (TSNC) architecture implements a fully centralized TSN network model based on the initial definitions in the IEEE 802.1Qcc amendment [22]. Figure 1 gives an overview of the system and the flow of actions and events. The Central Network Controller (CNC) module is responsible for configuring network elements and crucial services for the network operation, providing a high-level and uniform Application Programming Interface (API) for other services to interact with NEs in the network (i.e., routers, bridges, APs, end nodes). The Centralized User Configuration (CUC) offers interfaces for application and management-level interactions, such as requesting a network slice with certain performance characteristics, and registering or removing application definitions. Next, we describe the workflow of actions indicated by the numbers in Fig. 1 and which section details each element involved:

1. A network manager registers application definitions in the CUC, which are data models specifying the requirements and characteristics of network applications. Section describes the data models.
2. Applications are prepared to first request an application instance to the CUC, using a definition as a template. The CUC will analyze and trigger the necessary network configuration to admit the instance and deliver the expected network performance. The control plane modules involved in this action are described in Sect. , while the workflow for creating and removing application instances is presented in Sect. .
3. Based on the performance requirements and application characteristics specified in the definition, the CNC configures the elements that will transport the traffic of the application instance. This configuration includes applying INT rules to monitor the flows and setting up network slices through the network (detailed in Sect.).
4. The CUC notifies the application that the network is ready to transport packets of the application with the expected performance.
5. The CUC and CNC generate event messages about the creation of an application instance, as well as other relevant network events. Other micro-services, such as a

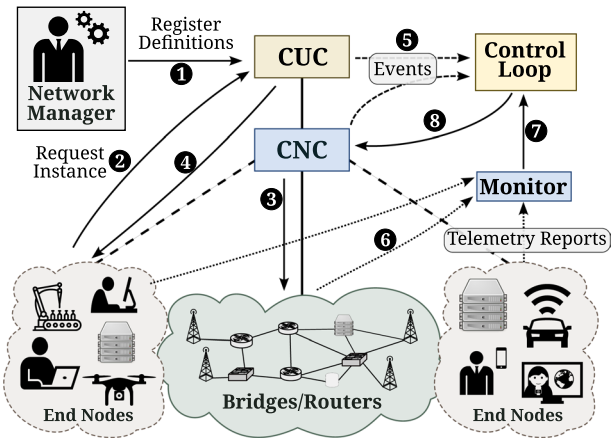

Fig. 1 Overview of the developed system

control loop, monitor these events to know which application instances to monitor and take corrective actions (if necessary) to maintain their required performance. These events are detailed in Sect. .

6. The bridges and other NEs publish telemetry reports to the Monitor element of the TSNC.
7. The Control Loop periodically collects telemetry reports from the monitor database and verifies whether the network performance of the monitored applications is within the expected performance bounds specified in the definition. Section details the operation of the control loop we implemented.
8. Lastly, if the control loop detects that the flows of an application instance are operating out of the performance bounds specified in the application definition, it takes corrective actions on the network via API calls to the CNC.

4 Control and monitoring enablers

This section gives an overview of the main enablers for the NS system, i.e., the SDN controller, the INT framework, and the traffic shapers implemented in a programmable data plane framework. For conciseness, we present a brief description of the main components and refer the interested reader to our previous works detailing the SDN TSN controller [7], INT framework [16], and software-based traffic shapers supporting a flexible selection of scheduling algorithms [17].

4.1 SDN TSN controller

Our control architecture uses a controller/agent model, appropriate for abstracting resource management in heterogeneous networks [23]. Figures 2, 3, and 4 show the diagrams of the TSNC and TSN Agent (TSNA) modules, and the communication flow between the CNC controller and TSNAs during network operation. The CNC is the central element for interaction with TSNAs, offering an API for other micro-services of the TSNC, such as a control loop, to get/set configurations on NEs. TSNAs are installed on NEs to enable fine-grained SDN control over their time synchronization, traffic shaping, switching/routing rules, and monitoring. These agents are not strictly required on end nodes; however, there would be a lower degree of control without the agent.

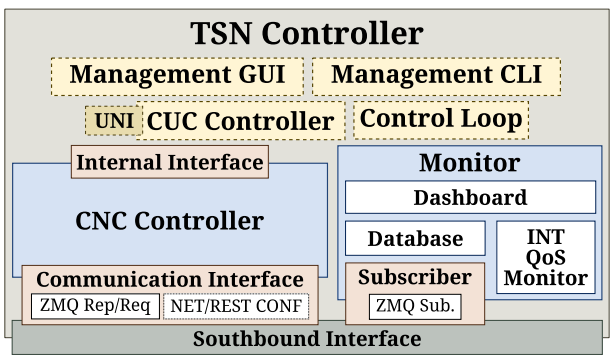

Fig. 2 TSN controller internal components

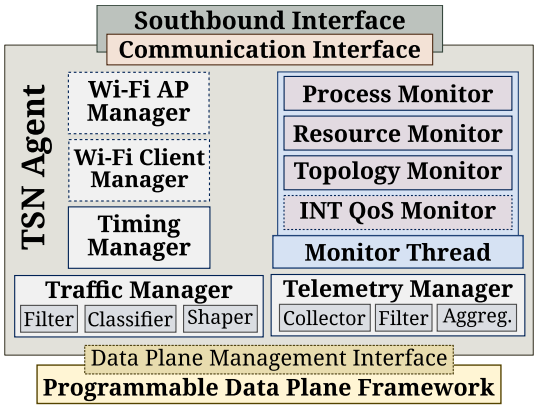

Fig. 3 TSN Agent internal components

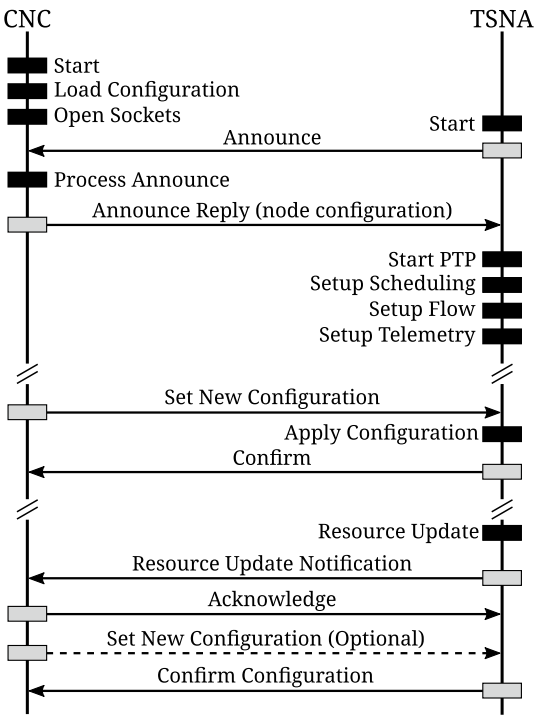

Fig. 4 CNC and TSNA communication workflow

The communication between CNC and TSNA, as well as between other micro-services of the controller, is implemented using ZeroMQ (ZMQ) messaging over TCP. The *Monitor* service receives telemetry reports from NEs and stores them in a long-term database, also providing a dashboard for data visualization. INT reports can be aggregated by the INT QoS Monitor to calculate throughput, delay, jitter, and PLR. The *Management GUI* offers a graphical interface for network operators to view and configure TSNC services and NEs, while the *Management CLI* offers a command-line interface for these operations. The CUC controller provides a User/Network Interface (UNI) for applications and network operators to inform flow specifications and requirements, so the network can automatically create, configure, and maintain network slices. Lastly,

the control loop monitors network flows and ensures that application instances operate within the expected KPIs bounds.

The TSNA is composed of submodules that control its operation according to the configuration received from the CNC (defined by the network operator). The Wi-Fi AP and Client managers are only active when the TSNA runs on such devices. The *Timing Manager* controls Precision Time Protocol (PTP) synchronization, and the *Traffic Manager* interprets and applies traffic filtering, classification, and shaping rules received from the controller. The *Telemetry Manager* collects, filters, and aggregates telemetry such as interface usage, PTP accuracy, and queue utilization. A separate thread of the TSNA constantly monitors relevant processes (e.g., for synchronization or Wi-Fi AP operation), resources (e.g., the status of network interfaces), or topology (detecting neighboring nodes joining or disconnecting). The monitor thread can also run the INT QoS Monitor, aggregating raw INT reports that contain only timestamps and counters and calculating QoS metrics of throughput, delay, jitter, and PLR. Depending on the data rate of a flow monitored by the INT framework, several INT reports might be generated every second, and aggregating these reports in the TSNA can reduce the amount of telemetry traffic generated in the control plane.

Figure 4 shows the communication flow between CNC and TSNA. The CNC starts by loading a network configuration file that defines the initial settings and authorized TSNAs, and then opens the southbound interface sockets. TSNAs start by announcing themselves to the CNC, informing the NE capabilities (interfaces, speeds, queues, timestamping), to which the CNC replies with the initial node configuration. Based on this reply, the TSNA sets up services in the NE for PTP synchronization, traffic scheduling, flow classification, and telemetry collection. At any point, the CNC might send new configurations to the TSNA, which are confirmed (whether successful or not) back to the CNC. When the TSNA detects a change in the NE resources, e.g., interface status, topology, and status of processes, these changes are immediately reported to the CNC. The CNC acknowledges these reports and might send a new configuration when applicable.

We extended the features of our controller with an event system that notifies micro-services, such as the control loop, about relevant events in the network. This event system is based on a ZMQ publisher socket, to which micro-services subscribe to get notifications of their interest. The CNC publishes events when a node connects, disconnects, or there is a topology change. The CUC publishes events when an application instance is created or removed (detailed in Sect.). The CUC was extended to operate with the data models introduced in Sect. , and trigger the appropriate API calls to the CNC when a new application instance is requested or terminates. Lastly, we developed the control loop service described in Sect. to monitor application instances and take corrective actions to maintain the expected KPI levels for the applications.

4.2 In-band network telemetry framework

The INT framework is a key part of our system, as it provides per-flow and per-hop real-time QoS monitoring. Figure 5 illustrates the system and its elements, which are integrated into our controller architecture previously described. The framework uses IPv6 extension headers to append metadata to regular data packets, which are then extracted

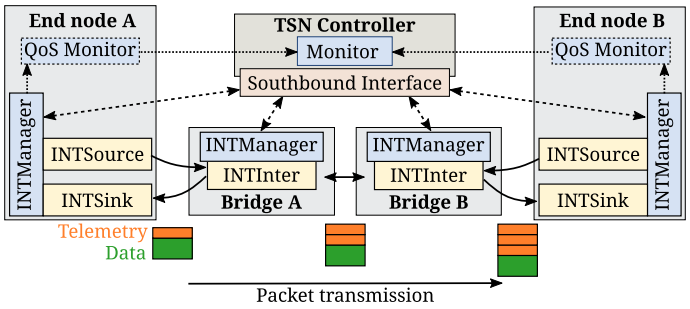

Fig. 5 INT framework overview

and combined to get network performance measurements.[1] The concept is illustrated in the bottom part of the figure, assuming a data packet is transmitted from end node A to end node B. The INTSource element starts the process by appending the initial extension headers and metadata (e.g., timestamps, packet counters) to the packet. The INTInter element at the bridges can append further metadata to the packet. The INTSink element extracts all the metadata and extension headers and processes this information to generate a report, then forwards the packets for further processing at the upper layers of the network stack.

The report generated by the INTSink contains values of counters and timestamps collected at each hop (referred to as raw INT report in the previous section). These reports can be aggregated by the QoS Monitor to calculate per-flow and per-hop QoS metrics. In our system, these QoS reports are published to the Monitor module of the TSNC. To minimize the telemetry overhead, we configure our system to collect only End-to-End (E2E) telemetry by default. The control loop reconfigures the INTInter elements to collect Hop-by-Hop (HbH) telemetry only when the E2E performance is not within the defined KPI bounds. Our INT framework can be flexibly reconfigured in runtime to support enabling/disabling HbH monitoring or changing the sampling rate (how often to append telemetry to data packets) for each flow. Therefore, for applications running on resource-constrained devices or that generate large amounts of traffic, the sampling rate of the INT framework can be tailored to balance accuracy and resource demands.

4.3 Programmable data plane structure

Our programmable data plane is based on the Click framework [24]. Figure 6 shows the logical structure of a NE, highlighting the programmable data plane framework. The elements of the Click framework are implemented in C++, and a configuration file describes how they are interconnected to form the packet processing pipeline. The solid arrows indicate the path of packets that ingress the system, and the dashed arrows show the path of packets being transmitted. In addition to the INT elements, we developed the *PrioToDevice* element that controls the buffering and transmission of packets according to the selected traffic shaping algorithm. The *ScheduleManager* provides the control plane configuration interface for the *PrioToDevice* element. As we use IPv6 extension headers for INT, we opted to use exclusively IPv6 for the data

[1] A complete description of the system and its performance evaluation can be found in reference [16].

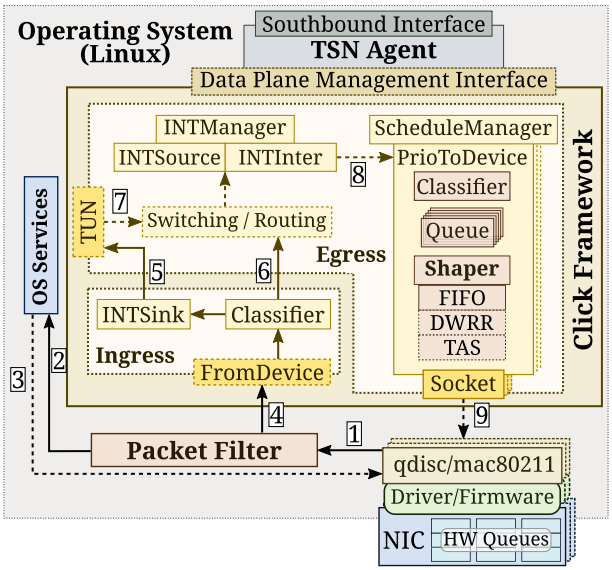

Fig. 6 Diagram of the elements and communication logic on NEs

plane (application data packets) and IPv4 for the control plane communication (e.g., control packets between CNC/TSNA, time synchronization packets).

Packets arriving at the NE pass through the *Packet Filter* of the Operating System (OS) indicated by arrow 1. Layer 2 and IPv4 packets belong strictly to control plane services in our architecture and are processed by their respective services in the OS and the TSNA (arrows 2 and 3). IPv6 traffic is directed to Click and received by the *FromDevice* element (arrow 4). Packets directed to the NE itself have their INT headers extracted and directed to a *tunnel* interface, to be processed by upper network layers (arrow 5). Packets egressing the NE (arrows 6 and 7) pass through switching and routing elements that decide the next transmission hop, then through the INTSource or INTInter elements, which define whether to add telemetry metadata to the packet or not. Next, the packet is directed to the *PrioToDevice* element associated with the selected egress network interface (arrow 8).

The *PrioToDevice* element has three main components: a classifier, a set of queues, and a set of traffic-shaping algorithms. The *classifier* defines in which queue to buffer the packets based on the Differentiated Services (DS) value in the packet header, while the *shaper* algorithm transmits the packet from each queue according to its configuration. We implemented three traffic-shaping algorithms: First-In, First-Out (FIFO), Deficit Weighted Round Robin (DWRR), and Time-Aware Shaper (TAS). FIFO is the default algorithm, using a single queue and simply transmitting the packets in their arrival order. DWRR [25] is a widely used algorithm that allocates bandwidth fairly among multiple queues based on their assigned weights. The values of the weights depend on the application priority and are defined by the control plane. Finally, the TAS algorithm is based on the IEEE 802.1Qbv mechanism, in which a Gate Control List (GCL) defines which gates to open at each moment of a time cycle. The scheduling algorithm selects a packet to be transmitted and sends it to the networking stack

of the system (arrow 9). We configure the queuing discipline in Linux to use FIFO to minimize the influence over the scheduling done in Click.

In this work, we create slices by allocating flows to specific queues and applying schedules to control the performance of each flow. The TAS is one of the main algorithms for traffic shaping for TSNs. However, the calculation of schedules using TAS is complex and might be computationally costly. Therefore, in our experiments, we have opted to use the DWRR algorithm, which offers a simpler but still effective way to control the performance of multiple flows. A single *weight* (or quantum) parameter assigned to each queue defines how many bytes of that queue can be transmitted on each round. The end-to-end performance of a flow is proportional to the weight allocated to the slice through the network path.

Due to the execution model Click and PrioToDevice, our implementation of the DWRR differs slightly from the classic algorithm but still achieves the same goal. We define a DWRR cycle of 100 μs, over which all queues must be served. The control plane must ensure that the sum of queue weights matches the capacity of the network adapter. For example, for a 1 Gbps link, the Network Interface Controller (NIC) can transmit 12,500 bytes per DWRR cycle, which is the sum of weights for a correct operation of the DWRR. At each round, the queue weight is added to the deficit counter associated with the queue. A queue can only be served if its counter is above 0. When a packet is transmitted, the queue deficit is decremented by the packet size in bytes. The deficit can accumulate and be used in the next round if the queue is not empty. If a queue is emptied during a round, its deficit is set to 0.

5 Data models and instances

The control plane needs information on application KPIs to make automated decisions on flow and resource allocation. Specifications of traffic characteristics also help to create schedules and define how to prioritize packets. To support this, we defined data models based on the concepts of IBN and ZSM by ETSI and 3GPP [10, 26]. Figure 7 shows how various elements are composed to specify application characteristics and network requirements. The data model is split into a generic *Data Model* for characteristics and requirements, and a *Concrete* part, generated at runtime when the application joins the network.

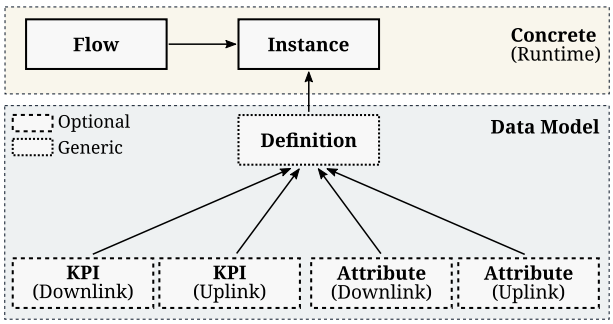

Fig. 7 Composition of an application instance

KPIs and *Attributes* are optional elements that are combined to create a generic *Definition*. *Definitions* are generic because they serve as a template for instances; thus, the same *Definition* can be used by several different application *Instances* active in the network. An *Instance* is composed of an application definition and the 5-tuple flow information of the endpoints exchanging traffic: Source/Destination IP, Source/Destination port, and transport protocol (TCP/UDP/ICMP).

5.1 Definitions

Definitions are stored and managed by the CUC, which offers an API for network operators and users to register, consult, or remove definitions. Our application definition is based on the *Intent* specification of the 3GPP TS 28.312 document [10], including similar fields derived from it. Table 1 describes the items. A unique *Label* identifies the definition and is used by applications or operators to request an instance. Next, there is a user-friendly *Description*, followed by the *Active* field, which indicates whether a definition can become an instance (useful to restrict sets of definitions based on the deployment). The *Priority* controls which applications have priority over others, for example, increasing the probability of their instances being admitted or having higher priority on resource allocation.

The *Observation Period* sets the interval for checking if the measured performance meets the KPIs, with a fulfillment report being generated every observation period. 3GPP defines the observation period in seconds; however, to support faster reaction times in industrial use cases, we opted to use milliseconds for this item. The *Actuation Period* defines the minimum wait time between consecutive actions affecting the application flows (e.g., adjusting slice resources), as some applications might take a few seconds to adjust to a new configuration. Lastly, the definition includes the KPIs and Attributes structures, specifying performance requirements and traffic characteristics of the application.

The KPIs are measurable performance metrics of a network flow, equivalent to *intentExpectations* from 3GPP specifications [10]. In our implementation, the *KPIs* define the QoS requirements of the application in terms of throughput, delay, jitter, and PLR. The KPIs are specified as the lower and upper bounds for throughput in Bytes/s, delay and jitter in milliseconds, and PLR in percentage. The interpretation of KPIs and how to

Table 1 Definition specification items

Name	Type	Description
Label	String	Identifier of application definition. *userLabel* in TS 28.312
Description	String	User-friendly description of this definition
Active	Boolean	If the definition is active and can become an instance. *intentAdminState* in TS 28.312
Priority	Integer	Priority of the application. *intentPriority* in TS 28.312
Observation Period	Integer	Interval in milliseconds between consecutive KPI fulfillment analysis. *observationPeriod* in TS 28.312
Actuation period	Integer	Interval in milliseconds between consecutive actions directly related to an instance of this definition
KPI	Structure	Performance requirements of the application
Attributes	Structure	Description of traffic characteristics of the application

correlate or compare them with the network measurements is a task carried out by the control loop service. *Attributes* describe the traffic generated by an application, such as frame sizes and packet generation intervals. If this information is known, it helps the control plane with admission control and scheduling. For instance, constant traffic with large frames might be impossible to admit, while an application that generates shorter frames in longer intervals can be admitted. *Attributes* specifications contain the following items: (i) *Transmission cycle* in milliseconds; (ii) *Frames per cycle*; and (iii) *Frame size* in bytes. Similar to KPIs, these items are specified as the upper and lower bounds of the values.

We represent all the structures in JSON, which is a concise and flexible format. Listing 1 shows an application definition example (the *data model* shown in Fig. 7) for an application that generates unidirectional traffic in uplink. The ranges of KPIs and attribute items are represented as lists with the lower and upper ranges. For fixed values, such as the *tx_cycle* in this example, the upper and lower bounds are equal, meaning that the application generates frames in fixed intervals of 5 milliseconds. For KPIs that do not have a strict requirement, the definition simply specifies a wide range, as for *throughput* in this example. Since the application only generates uplink traffic, the KPIs and Attributes structures for downlink are left empty.

```
"udp_app_basic": {
  "label": "udp_app_basic",
  "description": "One-way udp application",
  "active": true,
  "priority": 100,
  "actuation_period": 4000,
  "observation_period": 4000,
  "kpis": {
    "downlink": {},
    "uplink": {"throughput": [0, 10000000], "delay": [0, 10], "jitter": [0, 2],
          "loss": [0, 10]}
  },
  "attributes": {
    "downlink": {},
    "uplink": {
      "tx_cycle": [5, 5], "frames_per_cycle": [1, 10], "frame_size": [64, 512]
    }
  }
}
```

5.2 Application instances and their life cycle

Application instances are created based on *Definitions* and represent active instances of an application in the network. Table 2 describes the elements of an application instance, starting with a unique alphanumeric *Instance ID* and a numeric *Application ID* assigned

Table 2 Instance structure kept by the CUC

Name	Description
Instance ID	Unique identifier of an application instance (generated by the CUC)
Application ID	Numeric identifier of the instance (generated by the CUC) used for INT
Definition	Definition structure from which the instance was created
Flow	Structure with downlink and uplink 5-tuples
Slice	Structure defining the slices for uplink and downlink flows

by the CUC. This numeric identifier is used by services such as the INT framework, which uses integers to identify monitoring rules. Next, the *Definition* structure previously described is included, along with the *Flow* structure that specifies the endpoints running the application in downlink and uplink directions. Lastly, the *slice* structure stores the slices that the downlink and uplink flows are assigned. The downlink traffic of an application instance might be assigned to a different slice than the uplink traffic, supporting more flexible slice allocation schemes.

The CUC manages the life cycle of an application instance, listening to requests in the UNI and acting on the network through the CNC. The life cycle management of slices is carried out by the CNC, upon requests from the CUC or the Northbound (NB) Interface. Figure 8 describes the workflow for creating and terminating an application instance. Actions such as setting classification, INT, and scheduling rules are always confirmed by the TSNA to the CNC/CUC but were omitted in the figure. The diagram starts with the registration of an application *Definition*, which can be requested to the CUC via the UNI if a suitable definition for the application does not exist in the CUC yet. Then, an *Instance Request* call starts the instance creation process.

First, an admission check is done by the CUC, and the initial slice for the application is defined, depending on the network route, KPIs, and attributes. Application instances might share slices if the CUC determines that their requirements can be satisfied. The CUC may allocate the application to an existing slice, or request the creation of a new slice, which triggers the CNC to generate and apply scheduling rules in the network. The CUC makes calls to the CNC to apply flow classification rules in the nodes. In essence, the TSNAs uses *iptables* to mark the packets of the flow with a Differentiated Services Code Point (DSCP) value, which we later assign to a queue for scheduling. Next, the CUC gets the existing INT rules in the network and generates INT rules for the new

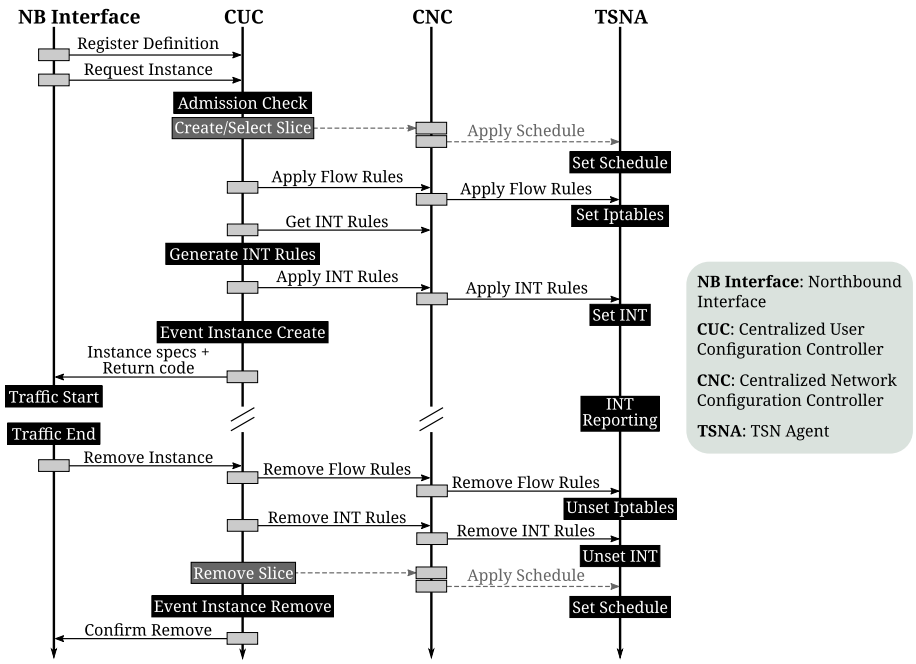

Fig. 8 Workflow for application instance creation

instance using an ID for the rule that does not overlap with any existing INT rules. The CUC triggers an event informing other services that a new instance was created and returns the specification and a success or failure code to the requester in the UNI. At this moment, the network is ready, the application can start running and transmitting packets, and the control loop can monitor if the network performance meets the expected KPIs.

During steady operation, the TSNAs transmit INT reports to the monitor module of the TSNC. These reports are stored in a database and can be consulted by other microservices like the control loop. When the application terminates, it issues a termination call to the CUC, which triggers other API calls to remove the classification and monitoring rules. If the instance was allocated to a dedicated slice, the CUC can request the slice removal to the CNC, which then applies new schedules in the network, removing the selected slice. An event is issued announcing the termination of the instance, and a confirmation is sent back to the application.

6 Control loop

During an application execution, the performance of its flows might fluctuate due to various circumstances, especially when wireless links are part of the end-to-end route. The *Control Loop* monitors the performance of the applications and adjusts the network slices to ensure that the KPIs requested by an instance are met. The control loop listens to events of the CUC and CNC. When a new instance is created, the control loop creates a monitor thread that constantly checks if the performance of the flows of an application is within the specified KPIs. This thread consumes telemetry reports from the *Monitor* database and compares the measurements against the KPI ranges. Figure 9 gives an overview of the flow of actions and events of the control loop when any of the KPIs are not fulfilled. A *Fulfillment Report* is generated every *observation period*, for each application instance. If a KPI is not fulfilled, the *Fulfillment Report* indicates which flow (uplink/downlink) and metric (throughput/delay/jitter/PLR) is not fulfilled, and an internal event is generated in the control loop.

The *Fulfillment Unmet* event triggers an analysis phase, which reads the telemetry data to identify which hops are the main cause of performance degradation. As mentioned in Sect. , the default setting of the INT framework is to collect only E2E telemetry to minimize overhead, thus, HbH telemetry is usually not immediately available. The analysis function determines whether HbH telemetry is available and enables it if necessary. For

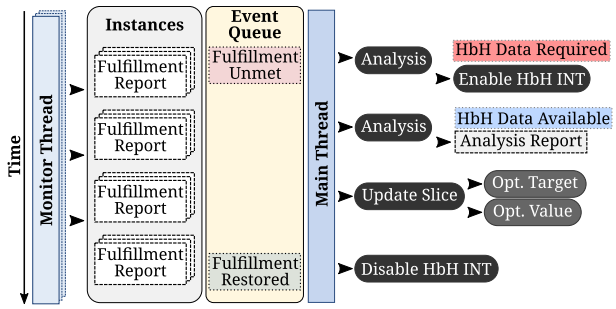

Fig. 9 Control loop flow overview

this step, the control loop gets the topology from the CNC, identifies the path of the flow, gets the INT rules monitoring the flows of the instance, and issues API calls to the CNC to enable HbH telemetry for the flows. When HbH telemetry becomes available, the analysis function generates a report identifying the main bottlenecks of the flows. This report has a structure indicating the hop (a tuple of source and destination node IDs) and the telemetry measurements. The hops are sorted according to the "bottleneck magnitude": e.g., if the E2E delay is 20 ms, and the delay on a given hop is 15 ms, this will be the first hop of the list in the analysis report. This way, corrective actions, such as a schedule update, can be planned focusing on the links that contribute the most to the performance degradation.

After getting a complete analysis report, the main thread of the control loop schedules a slice update. If multiple instances are currently in the analysis phase for unmet KPIs, the main thread waits for the conclusion of these analyses to call the *Update Slice* function. The update slice function first defines the optimization target, i.e., in which interface of which NE to take action. For this step, the analysis reports are verified to identify the bottleneck node and interface. Then, the second step is to define the optimization value, which depends on the traffic-shaping algorithm being used. Our current implementation focuses on slicing using the DWRR algorithm; therefore, the optimization value definition consists of selecting a new weight to be assigned to a given queue. With the optimization target and values defined, the *Update Slice* function issues a call to the CNC to update the queue weights. When the monitoring thread detects that the KPIs are again fulfilled, it waits 2× *Observation Period*, then generates a *Fulfillment Restored* event to the main thread. The main thread disables the HbH monitoring of the flows of the instance. We define this longer observation period to disable the HbH monitoring to ensure that the issue is solved and no further analysis is required.

7 Results and discussion

In this section, we evaluate the capability of our system to create and manage slices in wired and Wi-Fi networks. First, we create application definitions with different requirements, then show how the system can instantiate the end-to-end slices and provide the requested KPIs for applications wrapped to request a slice to the network. Each application instance is assigned to an exclusive slice. We compare this scenario with a network with no slicing capabilities, in which the applications are served in a best-effort manner. In the second part, we analyze the performance of the control loop, monitoring the flows in real-time and updating slice configurations to fulfill the requested KPIs. We evaluate the reaction time of the control loop upon detecting network issues and its capacity to apply the necessary corrections at the correct network bottlenecks.

Figures 10 and 11 show the wired and mixed (wired/Wi-Fi) topologies, respectively. The topologies were planned such that the flows we aim to control (flows 1–4) are not affected by the saturating traffic at their sources, but do converge in the access network (i.e., switches and AP), requiring the slicing mechanism to enforce traffic scheduling and classification rules to avoid their disruption by the saturating traffic (generated by node *tsn09)*. This is a common scenario in computer networks, where end nodes generate/receive a single or few data flows, while the access network (switches, routers, APs) serving several clients must deal with multiple clients contending for resources.

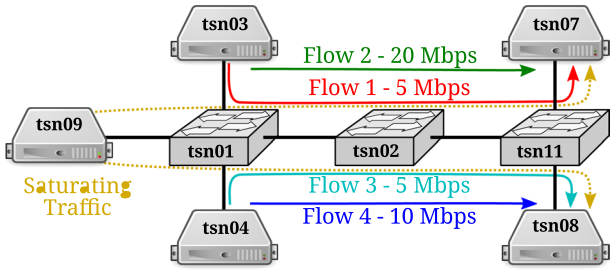

Fig. 10 Topology and flows used in the wired slicing experiments

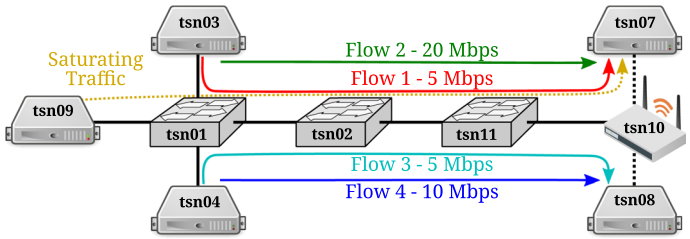

Fig. 11 Topology and flows used in the Wi-Fi slicing experiments

Table 3 KPIs for the flows in the slicing experiments

Flow ID	Goal	Minimum throughput	Maximum throughput
1 and 3	5 Mbps	4 Mbps (500000 B/s)	6 Mbps (750000 B/s)
2	20 Mbps	19 Mbps (2375000 B/s)	22 Mbps (2750000 B/s)
4	10 Mbps	9 Mbps (1125000 B/s)	12 Mbps (1500000 B/s)

Nodes *tsn01*, *tsn02*, and *tsn11* are industrial mini-pcs equipped with Intel i225-V Ethernet interfaces, Intel Celeron J4125 CPU, and 8GB of RAM, operating as switches. The other nodes are Intel NUC mini PCs. In the mixed topology of Fig. 11, *tsn10* operates as a Wi-Fi AP, while *tsn07* and *tsn08* are Wi-Fi stations, all of them using Sparklan WNFT-238AX Wi-Fi adapters. The figures indicate the flows generated between the different nodes. The application definitions define the throughput requirements for each flow, with the lower and upper bounds listed in Table 3. These flows are similar to video-streaming applications with different levels of resolutions and bitrates. The throughput of Flows 1 and 3 must range from 4 Mbps to 6 Mbps, flow 2 requires 19 Mbps to 22 Mbps, and flow 4 requires 9 Mbps to 12 Mbps. These flows are generated using a version of *iperf*[2] wrapped to first request an instance to the controller, as described in Sect. . The background traffic represents other flows running in the network, e.g., users in an office downloading documents and accessing web pages, competing for network resources with the other managed flows. The background traffic is generated using *iperf* with unrestricted UDP traffic.

[2] https://iperf.fr/.

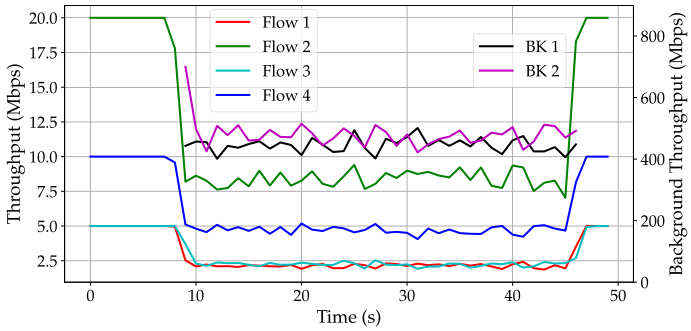

Fig. 12 Performance of the flows in a best-effort network (no slicing enabled)

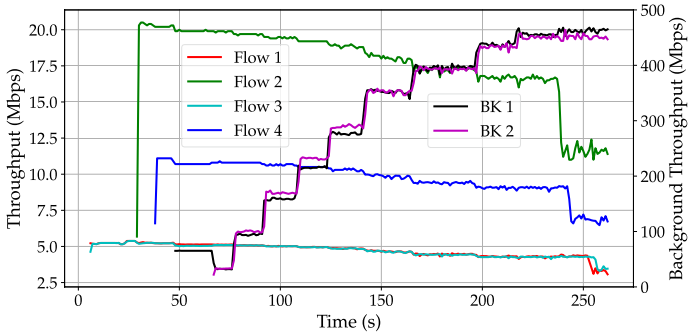

Fig. 13 Performance of the flows in the network with slicing enabled

7.1 Network slicing basic operation

Figure 12 shows the results for a wired network with no slicing capabilities enabled. Flows 1 to 4 have their throughput limited by the *iperf* tool, not by the network configuration. The background saturating flows (BK 1 and BK 2) are generated with *iperf* with unlimited throughput. From the start of the experiment, each *iperf* instance generates traffic with the configured throughput (values on the left y-axis), until at eight seconds in the test we start the background flows (values on the right y-axis). The four flows are severely affected by the saturating traffic, dropping to under half of their configured throughput, while the background flows reach around 450 Mbps. Flows 1–4, representing services such as surveillance, VoIP calls, or video-based inspections, might be disrupted by the background flows.

Figure 13 shows the results for a network with slicing enabled, where each flow is assigned to a queue, and the performance of each queue is defined by the weight allocation of the DWRR algorithm. Each *iperf* flow (except those generating background traffic) first requests an application instance to the CUC, which configures the network to deliver the specified performance. By default, the system sets a weight of 1000 for the best-effort flows, limiting them to approximately 75 Mbps. During the test, we gradually request the weight of the background flows to be increased by 1000, thus the steps observed in the throughput of the *BK 1* and *BK 2* flows. The results show that the system is capable of maintaining the requested performance of the slices for flows 1 to 4, but the performance starts degrading as we increase the resource allocation for the background flows. From the start of the experiment until approximately 120 s, all flows operate

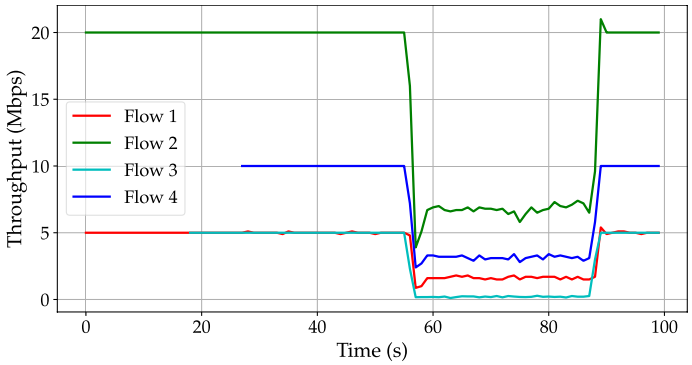

Fig. 14 Wi-Fi flows in best-effort network with saturating traffic

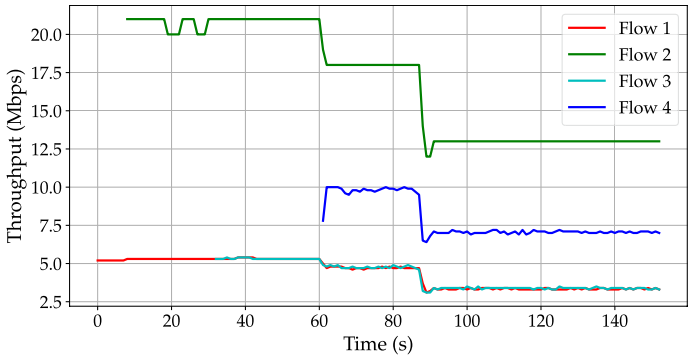

Fig. 15 Wi-Fi flows in the network with support for slicing

within the expected margins defined in Table 3. At 150 s in the experiment, with the background flows running over 300 Mbps each, flow 2 drops to approximately 18 Mbps, while flows 1, 3, and 4 are still within the expected margins. This is due to the higher load at the switches, as they need to process more packets per second.

The network reaches a saturation point at 220 s, with a slight increase in the background flows to approximately 450 Mbps. At this point, flows 1–4 can still maintain some of their required performance, especially if we compare these results with the network without slicing, where the throughput of the flows drops below half of the expected. Moreover, with the slicing-capable network, the throughput of each flow is controlled by the traffic shaping algorithm, and not by the transmission rate configured in *iperf*. At 240 s, the network is already saturated by all the traffic, and the increase in the best-effort queue weight by another 1000 surpasses the capacity of the link, thus, a steep performance drop in flows 1–4 is observed. This is a case where the control plane assigns a higher sum of weights than the network can sustain, thus, it is important that the CUC correctly performs the admission control of new flows. Still, the results show the feasibility of the system to handle multiple slices, delivering distinct performance levels, and background traffic.

Figures 14 and 15 show the results in the mixed topology. The saturating traffic in this test was generated from *tsn09* to *tsn07*, achieving a throughput of approximately 250 Mbps, omitted in the results for better visualization. Without slicing (Fig. 14),

there is a significant performance drop at around 55 s, when the saturating traffic is started. Flow 3 is practically interrupted, while Flow 2 drops from 20 Mbps to approximately 7 Mbps. When the saturating traffic is interrupted, the throughput of the flows returns to the original setting.

Figure 15 shows the result with the slicing functions enabled, in which the throughput of each flow is regulated by the traffic shaping system. When flow 4 is initiated at 60 s, it already causes a minor disruption to flows 1 and 3; however, they are still within the KPI range. Flow 2 suffers a stronger disruption and drops below the required minimum of 19 Mbps. At 70 s in the test, we start the saturating traffic and observe a throughput drop of all the managed flows. None of the flows gets completely interrupted, as occurred in the case without slicing, but the performance of the flows is still degraded below the expected KPIs. This behavior can be explained by the higher contention for the transmission medium caused by the competing flows generated to both Wi-Fi clients. Such contention requires adjustments to the resource allocation for the different flows, and this is a task for the control loop.

For the tests shown in Figs. 14 and 15, the control loop was not active, and introducing a fourth flow was already sufficient to disturb the operation of other flows. We executed an additional test run with the control loop enabled. The results in Fig. 16 show that when flow 4 is started, it causes the same disruption to the performance of other flows, with the throughput of flow 2 dropping below the minimum threshold. The control loop performs the analysis (enabling HbH monitoring, identifying the bottlenecks, and taking action) and adjusts the allocated weights for flow 2. At around 155 s in the test, flow 2 returns to operate within the expected throughput range defined in its KPIs.

The results show that the slicing mechanism can successfully enforce traffic differentiation over network flows, while still allowing background traffic to coexist. Moreover, the same software elements can be applied to wired and Wi-Fi networks, highlighting the flexibility of the system. However, under high levels of saturation in wired networks or in the case of Wi-Fi networks with the complexity of operating in a shared medium, the slicing mechanism alone is insufficient to guarantee the expected KPIs. Thus, a system to perform runtime adjustments in resource allocation is still necessary.

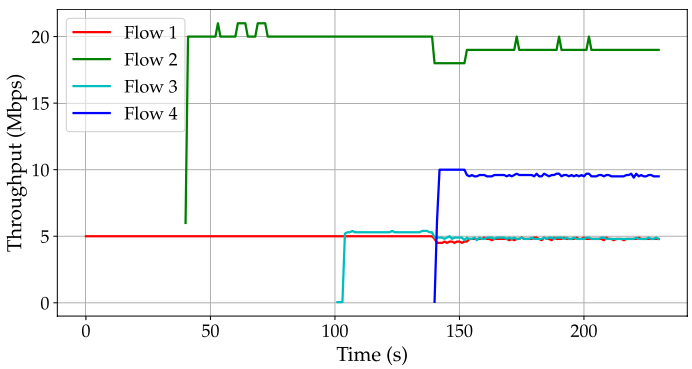

Fig. 16 Wi-Fi flows in the network with slicing and the control loop enabled

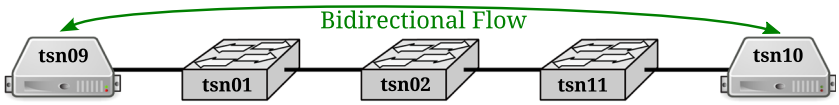

Fig. 17 Topology used to evaluate the control loop operation

Table 4 Downlink and uplink KPIs for the bidirectional flow used to evaluate the control loop operation

KPI	Minimum	Maximum
Throughput	1 Kbps (125 B/s)	80 Mbps (10000000 B/s)
Delay	0 ms	15 ms
Jitter	0 ms	2 ms
PLR	0 %	0.5 %

Table 5 Observation period and QoS report interval for the test cases

Test case	Observation period (ms)	INT QoS report interval (ms)
1	1000	1000
2	500	1000
3	100	1000
4	500	500
5	100	100

7.2 Control loop operation

We evaluated the reaction time of the control loop from detecting when an application is operating out of the expected KPIs, to analyzing the bottlenecks and taking corrective action. The goal is to evaluate the correct execution of the flow described in Fig. 9, including the on-demand activation of HbH INT monitoring, bottleneck analysis, and definition of NE and interface to take action. We also evaluated the effect of different observation periods on this reaction time, as it determines how often the monitoring threads verify the compliance of the KPIs. Further, we updated the INT framework and TSNC elements to support QoS reporting intervals as low as 10 milliseconds. Therefore, in addition to the observation period, we also evaluated using different INT QoS reporting intervals.

Figure 17 shows the topology for this test with two wired end nodes and three switches. We generated a bidirectional UDP flow between the end nodes with the KPIs described in Table 4.[3] The application definition does not have strict demands for throughput, defining the range of 1 Kbps to 80 Mbps for this metric. However, it requires delays under 15 ms and jitter under 2 ms. A PLR of up to 0.5 % is defined as acceptable for this application. We evaluated different configurations of the observation period in the application definition and INT QoS reporting intervals. Table 5 describes the test

[3] We based these requirements on VoIP requirements (https://www2.voipspear.com/posts/10), but due to the size of our PoC network, we reduced the upper limit of delay from 150 ms to 15 ms, and of jitter from 20 to 2 ms.

cases, starting from the baseline with one second for the observation period and QoS reporting. In test cases 2 and 3, we reduce the observation period but keep the default QoS report interval of one second. Finally, test cases 4 and 5 have matching observation periods, and QoS report intervals of 500 ms and 100 ms, respectively.

We executed the experiments by starting the application generating the bidirectional flow, and inserting a random delay at a random interface in the path of the flow. The link causing the issue might be one of the eight egress interfaces (4 on each downlink/uplink direction) of the topology from Fig. 17. Table 6 shows the times taken from the insertion of the network impairment until the recovery of the performance by an action of the control loop, in milliseconds. We executed five repetitions of each test case. The results show that reducing the observation period has a small effect, with the recovery time slightly reducing from 3.80 to 3.04 s on average. However, lowering the INT reporting interval and matching this interval with the observation period configuration significantly improved the response time of the control loop. Using 500 ms intervals, the time to recover was under 2.5 s, with a minimum of 521 ms. With 100 ms intervals, the reaction time was under 1 s on average, with a minimum value of 143 ms. For applications such as VoIP or Video-on-Demand, these recovery times might mean unnoticeable disruption to the users, while for critical industrial applications, they could be catastrophic. Nevertheless, these timings can be used by network-aware applications to define their operation in the face of transient network disruptions.

The results show the capacity of the control loop to quickly act in the network, despite the complex set of actions that need to be taken. Further improvements can still be explored to speed up the operation of the control loop for use cases requiring faster reactions, for example, keeping the INT HbH always active for highly sensitive flows, speeding up the analysis phase and identification of the bottlenecks affecting the flows.

8 Conclusion and future work

This paper presents a solution for QoS-driven automated network slice management over TSN networks. Building on recent advances in network telemetry and flexible network control enabled by INT and TSN standards, we describe a bottom-up approach for automated slice management inspired by ZSM and Intent-based Networking definitions. We describe the enablers that we have developed for network monitoring and control, as well as the data models that express the characteristics and requirements of network applications. These models provide the basis for the automated deployment of slices and their real-time monitoring by a control loop. We describe the steps to prepare the network for a new application instance and the steps that must be taken by a control loop

Table 6 Time (in milliseconds) for the control loop to detect the network issue

Test case	Mean	Minimum	Maximum	Std. Dev.
1 – (obs. 1000 ms, INT 1000 ms)	3800	3051	4108	475
2 – (obs. 500 ms, INT 1000 ms)	3159	3031	4069	320
3 – (obs. 100 ms, INT 1000 ms)	3048	1045	4099	825
4 – (obs. 500 ms, INT 500 ms)	2447	521	3733	949
5 – (obs. 100 ms, INT 100 ms)	994	143	1855	592

service to take corrective actions when the KPIs demanded by an application are not met by the network. These specifications may provide relevant insights to other researchers and developers working on automating network operations. The results from this PoC show the feasibility of the complete solution to automate the process of identifying and acting on the network to achieve dynamic and application-specific network troubleshooting.

In future works, we aim to evaluate the system over larger networks and with applications with more heterogeneous requirements, in order to study the scalability of the control loop. The developed components can also be used in emulated environments for comparison with other methods in the literature for quantitative comparison. In addition, we will extend the control loop to generate time-based schedules using the TAS algorithm, aiming to cover a wider range of use cases, including industrial applications. Lastly, we will integrate machine learning models to estimate QoE, introduced in our previous works [27], focusing on video on-demand applications, to provide a more holistic and user-focused operation.

9 Methods/experimental

In this work, we aim to demonstrate the feasibility of the solutions described through experimental results obtained using testbeds. The testbed is composed of computers of different configurations. Wired end nodes, Wi-Fi APs, and Wi-Fi clients are based on Intel NUC mini PCs, equipped with Intel i225-V and i219-LM Ethernet adapters, and Sparklan WNFT-238AX Wi-Fi adapters. The switches are industrial mini PCs with Intel Celeron J4125 CPU and Intel i225-V Ethernet adapters. The experiments were carried out using *iperf* traffic generator, generating UDP traffic with varied throughput configurations.

Abbreviations

3GPP	3rd Generation Partnership Project
AP	Access Point
API	Application Programming Interface
AVB	Audio Video Bridging
CNC	Central Network Controller
CUC	Centralized User Configuration
DS	Differentiated Services
DSCP	Differentiated Services Code Point
DWRR	Deficit Weighted Round Robin
E2E	End to end
ETSI	European Telecommunications Standards Institute
FIFO	First-In, First-Out
GCL	Gate Control List
HbH	Hop by Hop
IBN	Intent-Based Networking
IEEE	Institute of Electrical and Electronics Engineers
INT	In-band Network Telemetry
IoT	Internet of Things
IRTF	Internet Research Task Force
IT	Information Technology
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
ML	Machine Learning
NB	Northbound
NE	Network Element
NIC	Network Interface Controller
NS	Network slicing
OS	Operating System

OT	Operational Technology
PLR	Packet Loss Rate
PoC	Proof of Concept
PTP	Precision Time Protocol
QoE	Quality of Experience
QoS	Quality of Service
RSSI	Received Signal Strength Indicator
SDN	Software-Defined Networking
TAS	Time-Aware Shaper
TC	Traffic Control
TSN	Time-Sensitive Networking
TSNA	TSN Agent
TSNC	TSN Controller
UNI	User/Network Interface
URLLC	Ultra-Reliable Low-Latency Communication
ZMQ	ZeroMQ
ZSM	Zero-touch network and Service Management

Acknowledgements
This work has been performed within the European project SNS JU 6 G-Xcel (Grant Agreement No. 101139194). This research is also partially supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES)—Finance Code 001, São Paulo Research Foundation (FAPESP) with Brazilian Internet Steering Committee (CGI.br), grants 2018/23097-3 and 2020/05182-3, Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Author contributions
GMJ developed the software components, executed the experiments and data analysis, and prepared the manuscript. NSK contributed to the conceptualization of the data models, workflows, and review of the manuscript. JMB contributed with the organization of the manuscript and review of the content. DM contributed with the analysis and interpretation of the results, review, and organization of the manuscript.

Funding
This work has been performed within the European project SNS JU 6 G-Xcel (Grant Agreement No. 101139194). This research is also partially supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES)—Finance Code 001, São Paulo Research Foundation (FAPESP) with Brazilian Internet Steering Committee (CGI.br), grants 2018/23097-3 and 2020/05182-3, Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG), and Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Availability of data and materials
The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

Declarations

Competing interest
The authors declare that they have no conflict of interest.

Received: 31 October 2024 Accepted: 23 May 2025
Published: 18 December 2025

References

1. J. Haxhibeqiri, A. Seferagic, R.V. Bhat, I. Moerman, J. Hoebeke, Tighter application-network interfacing to drive innovation in networked systems. In: Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration. NAI'21, pp. 53–57. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3472727.3472801
2. L. Zanzi, V. Sciancalepore, A. Garcia-Saavedra, X. Costa-Pérez, OVNES: Demonstrating 5G network slicing overbooking on real deployments. In: INFOCOM 2018—IEEE Conference on Computer Communications Workshops, pp. 1–2 (2018). https://doi.org/10.1109/INFCOMW.2018.8406867
3. P.H. Isolani, N. Cardona, C. Donato, J. Marquez-Barja, L.Z. Granville, S. Latré, SDN-based Slice Orchestration and MAC Management for QoS delivery in IEEE 802.11 Networks. In: 2019 Sixth International Conference on Software Defined Systems (SDS), pp. 260–265 (2019). https://doi.org/10.1109/SDS.2019.8768642
4. K. Samdanis, X. Costa-Perez, V. Sciancalepore, From network sharing to multi-tenancy: The 5G network slice broker. IEEE (2016). https://doi.org/10.1109/MCOM.2016.7514161
5. L. Lo Bello, W. Steiner, A Perspective on IEEE time-sensitive networking for industrial communication and automation systems. Proc. IEEE **107**(6), 1094–1120 (2019). https://doi.org/10.1109/JPROC.2019.2905334
6. N. Finn, Introduction to time-sensitive networking. IEEE Commun. Standards Mag. **2**(2), 22–28 (2018). https://doi.org/10.1109/MCOMSTD.2018.1700076

7. G. Miranda, E. Municio, J. Haxhibeqiri, J. Hoebeke, I. Moerman, J.M. Marquez-Barja, Enabling time-sensitive network management over multi-domain wired/wi-fi networks. IEEE Trans. Netw. Service Manag. (2023). https://doi.org/10.1109/TNSM.2023.3274590
8. T. Zhang, G. Wang, C. Xue, J. Wang, M. Nixon, S. Han, Time-sensitive networking (tsn) for industrial automation: Current advances and future directions. ACM Comput. Surv. (2024). https://doi.org/10.1145/3695248. Just Accepted
9. A. Clemm, L. Ciavaglia, L.Z. Granville, J. Tantsura, Intent-based networking: concepts and definitions. RFC Editor (2022). https://doi.org/10.17487/RFC9315
10. 3GPP: Management and orchestration; Intent driven management services for mobile networks. Technical Specification (TS) 28.312, 3rd Generation Partnership Project (3GPP) (2023). Version 18.1.1. https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3554
11. B.K. Saha, D. Tandur, L. Haab, L. Podleski, Intent-based networks: An industrial perspective. In: Proceedings of the 1st International Workshop on Future Industrial Communication Networks. FICN '18, pp. 35–40. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3243318.3243324
12. K. Mehmood, D. Palma, K. Kralevska, Mission-critical public safety networking: an intent-driven service orchestration perspective. In: 2022 IEEE 8th International conference on network softwarization (NetSoft), pp. 37–42 (2022). https://doi.org/10.1109/NetSoft54395.2022.9844105
13. W. Cerroni, C. Buratti, S. Cerboni, G. Davoli, C. Contoli, F. Foresta, F. Callegati, R. Verdone, Intent-based management and orchestration of heterogeneous openflow/IoT SDN domains. In: 2017 IEEE Conference on network softwarization (NetSoft), pp. 1–9 (2017). https://doi.org/10.1109/NETSOFT.2017.8004109
14. C.E. Rothenberg, D.A. Lachos Perez,N.F. Sousa, R.V. Rosa, R.U. Mustafa, M.T. Islam, P.H. Gomes, Intent-based control loop for DASH video service assurance using ML-based Edge QoE estimation. In: 2020 6th IEEE conference on network softwarization (NetSoft), pp. 353–355 (2020). https://doi.org/10.1109/NetSoft48620.2020.9165375
15. S.K. Perepu, J.P. Martins, R.S. S, K. Dey, Intent-based multi-agent reinforcement learning for service assurance in cellular networks. In: GLOBECOM 2022–2022 IEEE Global communications conference, pp. 2879–2884 (2022). https://doi.org/10.1109/GLOBECOM48099.2022.10001426
16. G. Miranda, J. Haxhibeqiri, J. Hoebeke, I. Moerman, D.F. Macedo, J.M. Marquez-Barja, A Flexible in-band network telemetry framework for heterogeneous private networks. In: 2024 IEEE 20th international conference on factory communication Systems (WFCS), pp. 1–8 (2024). https://doi.org/10.1109/WFCS60972.2024.10540842
17. G. Miranda, J. Haxhibeqiri, J. Hoebeke, I. Moerman, D.F. Macedo, J.M. Marquez-Barja, Flexible wired/wi-fi tsn networking through sdn and soft traffic control. In: 2023 IEEE conference on network function virtualization and software defined networks (NFV-SDN), pp. 178–179 (2023). https://doi.org/10.1109/NFV-SDN59219.2023.10329729
18. A. Leivadeas, M. Falkner, A survey on intent-based networking. IEEE Commun. Surv. Tutor. **25**(1), 625–655 (2023). https://doi.org/10.1109/COMST.2022.3215919
19. E. Coronado, S.N. Khan, R. Riggio, 5G-EmPOWER: a software-defined networking platform for 5G radio access networks. IEEE Trans. Netw. Serv. Manag. **16**(2), 715–728 (2019). https://doi.org/10.1109/tnsm.2019.2908675
20. M. Richart, J. Baliosian, J. Serrat, J.-L. Gorricho, R. Agüero, Slicing in WiFi networks through airtime-based resource allocation. J. Netw. Syst. Manag. **27**(3), 784–814 (2019). https://doi.org/10.1007/s10922-018-9484-x
21. F. Fami, C. Pham, K.-K. Nguyen, Towards iot slicing for centralized wlans in enterprise networks. In: 2020 International symposium on networks, computers and communications (ISNCC), pp. 1–6 (2020). https://doi.org/10.1109/ISNCC49221.2020.9297339
22. I. Electrical, E. Engineers, IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: stream reservation protocol (srp) enhancements and performance improvements. IEEE Std 802.1Qcc-2018, 1–208 (2018) https://doi.org/10.1109/IEEESTD.2018.8514112
23. S. Schriegel, T. Kobzan, J. Jasperneite, Investigation on a distributed SDN control plane architecture for heterogeneous time sensitive networks. IEEE International workshop on factory communication systems - proceedings, WFCS **2018-June**, 1–10 (2018) https://doi.org/10.1109/WFCS.2018.8402356
24. E. Kohler, R. Morris, B. Chen, J. Jannotti, M.F. Kaashoek, The Click modular router. ACM Trans. Comput. Syst. **18**(3), 263–297 (2000). https://doi.org/10.1145/354871.354874
25. M. Shreedhar, G. Varghese, Efficient fair queuing using deficit round-robin. IEEE/ACM Trans. Networking **4**(3), 375–385 (1996). https://doi.org/10.1109/90.502236
26. ETSI: zero-touch network and service management (ZSM); Intent-driven Autonomous Networks; Generic Aspects. ETSI Doc. Number: GR ZSM 011 (2023). https://portal.etsi.org/webapp/workprogram/Report_WorkItem.asp?WKI_ID=67446
27. G. Miranda, D.F. Macedo, J.M. Marquez-Barja, Estimating video on demand QoE from network QoS through ICMP probes. IEEE Trans. Netw. Serv. Manag. **19**(2), 1890–1902 (2022). https://doi.org/10.1109/TNSM.2021.3129610

Publisher's Note