Enabling Time-Sensitive Network Management Over Multi-Domain Wired/Wi-Fi Networks

Gilson Miranda Jr.*†, Esteban Municio*§, Jetmir Haxhibeqiri‡,
Jeroen Hoebeke‡, Ingrid Moerman‡, Johann M. Marquez-Barja*
*University of Antwerp - imec, IDLab, Faculty of Applied Engineering - Antwerp, Belgium
†Universidade Federal de Minas Gerais - Computer Science Department - Minas Gerais, Brazil
‡Ghent University - imec, IDLab, Department of Information Technology - Ghent, Belgium
§i2CAT Foundation - Barcelona, Spain

***Abstract*—Deterministic performance and reliable operation are vital for many applications with industrial-grade requirements. Such applications rely on Time-Sensitive Networking (TSN) to enable time-critical deterministic communication. While standardization efforts were focused mainly on TSN features for wired domains, recent advances in wireless technologies (e.g., Wi-Fi 6/7) are extending time-sensitive communication towards wireless networks as well. However, achieving multi-domain LAN/Wireless LAN (WLAN) end-to-end TSN communication requires addressing challenges on end-to-end time synchronization, multi-domain control plane interoperability, run-time end-to-end scheduling, and fine-grained monitoring. Because state-of-the-art TSN controllers' scope lays far below these new required capabilities, in this work we present a novel, fully-programmable controller for end-to-end TSN-enabled networks. Our controller is based on a modular architecture to be adaptable to challenges arising when shifting the standard TSN scope towards WLAN domain. We deploy a proof-of-concept in a cloud-wired environment to evaluate its key performance indicators when handling increasing numbers of nodes and simultaneous requests. Further, we run experiments on real TSN-enabled networks comprising Ethernet and Wi-Fi technologies, demonstrating the effectiveness of the controller in performing seamless fine-grained traffic control in both domains.**

***Index Terms*—TSN, 802.1, SDN**

I. Introduction

Applications with industrial-grade requirements demand reliable and deterministic network performance. Ethernet has been used for industrial applications since the early days of the technology invention and solutions usually required over-provisioning, isolation of networks for specific tasks, and detailed traffic engineering [1]. These networks still suffered from many drawbacks such as high cost, unpredictability due to corner cases, and lack of flexibility. With origins in the audio/video context, the IEEE 802.1 Time-Sensitive Networking (TSN) Task Group has been developing a set of standards to enable reliable and deterministic communication over IEEE 802.1 networks [2], [3]. The new standards enable more reliable network operation for time-sensitive traffic and allow their coexistence with best-effort traffic on the same network.

As more features are incorporated and standardized for wired TSNs, the interest in extending those capabilities to the wireless domain also increases [4]. The flexibility allowed by wireless communication makes it attractive for industrial applications requiring mobility or more flexible deployments. However, the unpredictability of the wireless medium has been a major limitation to the wide adoption of wireless technology for deterministic applications. Still, Wireless TSNs (W-TSNs) are becoming a reality with recent developments, especially with those related to 5G technologies [5], and with Wi-Fi-based technologies using unlicensed bands [6], [7]. Besides industrial applications, W-TSNs also brings appealing capabilities for real-time applications like gaming and augmented/virtual reality (AR/VR) [4].

To accelerate the integration of W-TSN along with current IEEE 802.1 TSNs and the ecosystem around them, seamless operation and interoperability from LAN to WLAN domains are required [4]. In this sense, Software Defined Networking (SDN) control architectures for TSN must also take into account the existence of wireless components in the network. Additionally, in view of ongoing feature developments, it is interesting to have an extensible and flexible architecture that supports the fast integration of new features.

In this work, we present a modular, multi-domain controller architecture to provide end-to-end TSN-enabled control over LAN and WLAN domains. We identify the required building blocks to support multi-protocol conversion, QoS mapping, and cross-domain scheduling. Our controller architecture aims to perform such tasks in a flexible manner, allowing the use of different types of schedulers or control-loop algorithms according to use case demands, as well as extensions to support upcoming TSN features under development/standardization. Additionally, to coordinate and exert such operations in the network nodes, we specify an agent entity that holistically manages devices in near real-time for both LAN and WLAN domains. To show the feasibility of our controller, we present a proof-of-concept capable to initialize, monitor, and reconfigure an end-to-end TSN network comprised of wired Ethernet and Wi-Fi segments. We performed experiments in testbeds comprising a wired TSN, as well as an end-to-end W-TSN implemented on top of openwifi[1], an open-source implementation of Wi-Fi with support for TSN extensions [7], [8]. The results demonstrate the relevance of our multi-domain TSN controller to unveil the full potential of TSN networks.

This paper is organized as follows: Section II provides back-

[1]https://github.com/open-sdr/openwifi

ground about TSNs and the main related standards. Section III presents related work, focusing on SDN controller architectures for TSNs. In Section IV, we present our controller architecture, and in Section V we detail the agent module that is placed on network nodes. Section VI describes the setup of the experiments to evaluate the solution, and Section VII presents and discusses the obtained results. Finally, Section VIII concludes this paper. We provide a list of acronyms used in this paper at the end of the document.

II. Background

TSN originated from the multimedia domain with the Audio Video Bridging (AVB) Task Group by IEEE 802.1 in 2007 [1]. AVB brought improvements to IEEE 802.1 networks, but more mechanisms were still necessary to support the requirements of automotive and industrial applications [9]. The current TSN standards can be organized into four main pillars [9]: i) timing and synchronization; ii) bounded low latency; iii) reliability; iv) resource management.

As time-sensitive applications frequently operate based on strict deadlines, time synchronization is essential in many use cases. In addition, time-based scheduling, as proposed by IEEE 802.1Qbv relies on strict synchronization of the network for an effective operation. On wired TSN the Precision Time Protocol (PTP) enables time synchronization of nodes with nanosecond precision [10], [11]. Recently, the functionalities required by PTP were implemented and demonstrated over Wi-Fi using openwifi [8], [12], [13].

Standards IEEE 802.1Qav, Qbv, and Qbu introduce features for bounded low-latency communication [14]–[16]. The first two specify traffic shaping mechanisms that allow precise coordination of data transmission. Based on such features, network devices can be properly configured in order to provide the required performance to different traffic flows. The last standard defines a frame preemption mechanism allowing the interruption of an ongoing frame transmission in favor of frames with higher priority.

Addressing reliability in TSN, we highlight IEEE 802.1Qci and IEEE 802.1CB standards [17], [18]. The first provides enhancements for stream[2] filtering and policing to, for example, direct traffic to specific queues. IEEE 802.1CB defines frame replication and elimination, enabling packet transmission via multiple paths and its re-combination closer to the destination, discarding duplicates, and reducing the packet loss probability.

Addressing resource management, we highlight IEEE 802.1Qcc [19]. This amendment defines protocols for the configuration of stream reservations and stream requirements. The configuration process of a TSN network begins when Talkers and Listeners inform their requirements to the network, by means of a User-Network Interface (UNI). The UNI allows Listeners and Talkers to inform their application requirements (e.g., required QoS and stream characteristics) to the network elements (bridges or controllers), as well as to confirm whether they are ready to receive packets from a given stream. Then, there is a process of configuration of TSN features on Bridges along a tree from each Talker to the corresponding Listener(s). The document describes three TSN configuration models:

1) **Fully distributed:** in this model there is a UNI between the end node (talker/listener) and the bridge it is connected to. The configuration of each bridge in the path between the end nodes is performed locally, without a centralized network controller entity. The knowledge of each bridge does not necessarily include information about the whole network or the whole path.
2) **Centralized network/distributed user:** this model also provides a UNI between end nodes and the bridges they are connected to, however, the configuration of the bridges is carried by a centralized controller. The Centralized Network Configuration (CNC) has knowledge about network topology and capabilities of bridges, being able to define configurations that would not be possible using the fully distributed model.
3) **Fully centralized:** this model defines an additional entity, the Centralized User Configuration (CUC), which centralizes the configuration of Talkers/Listeners. The UNI is provided by a centralized configuration entity also encompassing CNC and CUC. This model aims at use cases that require significant configuration in the end stations, for example, cases in which the TSN streams must align with I/O timing of sensors/actuators [19].

This brief overview shows how many software and hardware components must be properly configured and coordinated to achieve a reliable and functional TSN network. With standardization still in process and new features envisioned for the future, the capacity to quickly integrate the new functions into existing TSNs can result in competitive advantages. The architecture that we propose and implement in this work supports existing functionalities from current standards for time synchronization, traffic shaping, and network coordination required by Ethernet-based TSN networks. Such features were extended to Wi-Fi in previous works [13], [20], and can be seamlessly managed by our controller.

In the next section, we analyze and discuss works in the literature that focus on the design, development, and evaluation of solutions related to the management of TSN networks through centralized network control.

III. Related Work

Different works such as [21], [30] have investigated the instantiation of general-purpose wired TSN in industrial contexts through an SDN controller. Although these works aim to ensure the dynamic configuration of TSN networks, some of their modeling and configuration services are still done offline, requiring manual re-execution every time the network conditions change (e.g., topology changes, device failures, etc.). This significantly increases the Time-to-Integrate and, thus, the deployment cost. In this line, there are also other control solutions to integrate wired TSN with industrial standards, such as OPC UA [22], [31], or PROFINET [23]. In contrast to these approaches, in the present work, we provide a fully dynamic configuration through the use of a closed-loop cycle of monitor and control.

[2]We use the terms “stream” and “flow” interchangeably in this work.

TABLE I: Summary of related work

Reference	Model	Wireless Support	Sync	Monitoring	Scheduling	Status
This work	Centralized/Hybrid	Yes, Wi-Fi	802.1AS, IEEE1588 (e2e)	Yes (e2e)	Yes (e2e)	Prototype. See Sec.VI
Said et al. [21]	Centralized	No	802.1AS	No	No	Prototype
Kobzan et al. [22]	Centralized	No	802.1AS	No	No	Prototype
PROFINET [23]	Centralized	No	PROFINET	No	No	Commercial
Nsiah et al. [24]	Centralized	Yes, DECT (not e2e)	DECT (not e2e)	No	No	Prototype (partial)
5G-TSSDN [25], [26]	Centralized	Yes, 5G (not e2e)	802.1AS (not e2e)	No	No	Prototype
Gutiérrez et al. [27]	Centralized	No	N/A	No	Yes	Model
Pop et al. [28]	Centralized	No	N/A	No	Yes	Model
Pahlevan et al. [29]	Centralized	No	N/A	No	Yes	Simulation

While these previous works are limited to wired domains, other works try to extend the ideas of TSN to wireless domains, such as Nsiah et al. [24]. The work introduces the concept for a TSN controller supporting wireless devices, specifically, Digital Enhanced Cordless Telecommunication (DECT) Ultra-Low Energy (ULE) [32]. However, these wireless extensions are used as uncoupled extensions of the wired TSN domain and do not offer actual end-to-end support (e.g., [24] it only demonstrates a few DECT-dependent operations performed through NETCONF). Other more mature works are the ones proposing the integration of traditional TSN wired networks with the Ultra-Reliable Low-Latency Communication (URLLC) slice of 5G networks, such as [25], [26]. Such works are based on a TSN bridge to integrating both domains, allowing a Time Sensitive SDN (TSSDN) controller [33] to allocate resources in the wireless domain. However the synchronization is still not end-to-end [20], and the proposed controllers do not support monitoring or scheduling.

Aligned with these works (and also with the same problem), there is also the work carried out in the IETF DETerministic NETworks (DetNet) working group (WG) focusing on the integration with 5G [34]. There are other works that tackle the dynamic control and optimal scheduling in TSN networks through optimization models, heuristics, and simulations [27]–[29], [35]. These theoretical approaches, however, are focused only on wired TSN and do not consider practical limitations such as the ones related to synchronization or monitoring.

A summary of the related works is presented in Table I. The comparative table describes the reference, the configuration model supported, whether the solution supports wireless devices or not, and the TSN components managed (i.e., synchronization, scheduling, or monitoring). The last column indicates whether and how each solution was evaluated. Notice that all related works target fully centralized configuration models, while our architecture additionally supports the centralized network/distributed user model. This is because clients are able to request flows to the controller through an In-band signaling protocol [36].

According to the existing literature, our work is the first in demonstrating a standalone TSN controller solution for both Ethernet and Wi-Fi. In addition, we propose a modular organization of the SDN controller for TSN in which each module is specialized in a specific feature set of TSN in order to fulfill the requirements for deploying effective TSN networks. This modular approach allows faster integration of new technologies, as well as support for different types of devices.

IV. TSN Controller Architecture

Before describing the TSNC components and the agents we define the network model considered in this work. We distinguish the Network Elements (NEs) in the following types: i) Controller; ii) Switches; iii) Routers; iv) Wired Nodes; v) Wireless Access Points; and vi) Wireless Clients.

Figure 1 illustrates a high-level view of the network model. The central element is the TSN Controller (TSNC), running the *CNC* module that governs node configurations by interacting with a *TSN Agent (TSNA)* deployed on the nodes. Controller/Agent architectures are useful for abstracting resource management on heterogeneous networks, being an appropriate choice for the context of Ethernet/Wi-Fi TSNs [37]. NEs can be a Router (A), Switches (B to E), Wireless Access Points (F), Wired Nodes (G and H), or Wireless Clients (I and J). If a NE requires fine-grained control (e.g., wired node G in the figure), it must run an instance of the TSNA to allow central coordination of PTP synchronization and scheduling. Otherwise, as in the case of wired node H, if PTP synchronization and scheduling are supported, they are not managed by the TSNC, and any TSN-like control over the traffic is done from node C onward. The WLAN domain is based on openwifi TSN features [7], [8], [13].

CNC and TSNAs communicate through a logical control network. The control plane uses IPv4 and the data plane uses IPv6, as we use the IPv6 extension headers for In-band Network Telemetry (INT) (detailed in Section V). Management can be performed using different configuration protocols, e.g., NETCONF, RESTCONF, or a custom protocol (detailed in Section IV-A). Message exchanges can be initiated by either TSNC or TSNA, therefore, bidirectional sockets are required. This approach allows TSNAs to immediately inform the CNC when changes occur in link connectivity, data rate, or any other relevant information, so the appropriate actions can be taken, for example, by a control loop module. Different from other approaches in the literature, the bidirectional connection brings benefits such as reduced control overhead by avoiding periodic polling, and faster detection of anomalies or topology changes. The capacity to quickly notify the CNC about relevant events is crucial for supporting TSN over Wi-Fi, as handovers might require fast adaptation of schedules and other configurations.

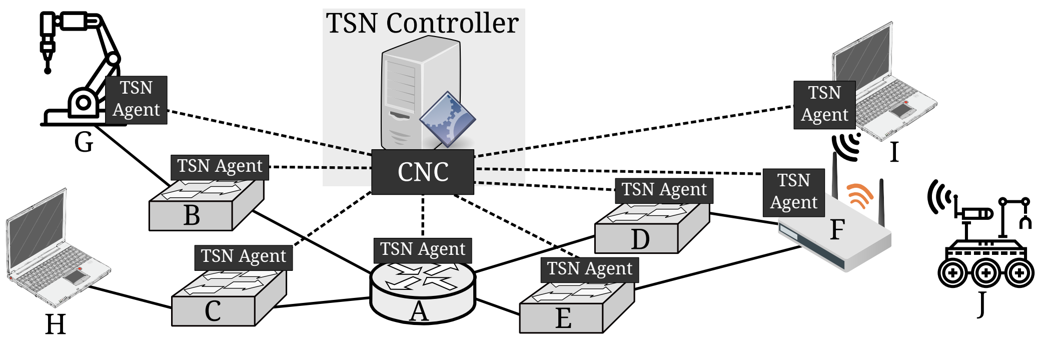


Fig. 1: Network model considered in this work

A. *TSNC Overview*

Figure 2 shows the internal modules of the TSNC. The base architecture is composed of the *CNC*, the *Monitor*, and the *Management Interface*. The other modules provide additional functionality that might be included as plug-ins depending on the deployment requirements. These modules are: the *Scheduler* for dynamic configuration of schedules; a *Control Loop* for automated real-time management; and a *CUC* for centralized user configuration and high-level network orchestration.

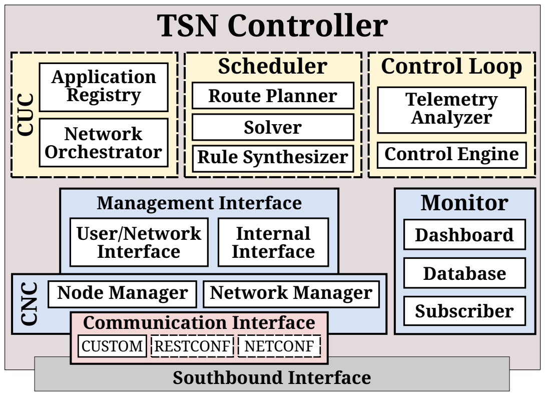


Fig. 2: TSNC Architecture

We define an abstraction layer for communication between CNC and TSNAs, referred to as *Communication Interface* in Figure 2. Although the current trend is to adopt Network Configuration (NETCONF) protocol for configuration, along with Yet Another Next Generation (YANG) data models [38], [39], this modular architecture gives the flexibility for different protocols, such as proprietary or in-house developed. When a configuration has to be applied or retrieved from a NE, the selection of the appropriate communication protocol (i.e., NETCONF, RESTCONF, or a custom driver) for each specific TSNA is abstracted by the *Communication Interface* component. This allows multiple protocols to be transparently supported by the CNC. Regardless of the communication protocol (or *Driver*, as referred to in this architecture), our design requires that both endpoints (CNC and TSNA) must be able to initiate transmissions.

The next subsections detail the TSNC modules. For objectivity, we describe them based on our implementation. For example, we rely on INT monitoring stack [40], [41] and Wi-Fi scheduling implemented in openwifi [7]. Internal communication between the TSNC modules uses the ZeroMQ (ZMQ)[3] messaging library which is available for over 20 programming languages, supports a variety of communication patterns (e.g., Publish/Subscribe, Request/Reply), and shows high performance with small messages such as the ones generated by the TSNC modules [42]. We also use ZMQ to implement the *Custom Driver* for communication between TSNC and TSNAs. We encode the messages in JavaScript Object Notation (JSON) in our custom driver, instead of XML used by NETCONF. JSON has lower overhead in comparison to XML, which results in lower network usage when messages are exchanged between CNC and TSNAs.

B. *Centralized Network Configuration – CNC*

The CNC centralizes the operations on the TSNC in terms of node and network management. The functions are split into two classes: node management and network management. As described in Section II, many software components must be carefully coordinated on all nodes of a TSN network, therefore, the operation of these processes on each node is configured and tracked by the *Node Manager* of the CNC. The node manager keeps track of the processes for PTP synchronization, INT, and monitoring of interface states of a node, as well as the current configuration state of a node. If the node is restarted, the node manager is in charge of restoring the last valid configuration of the node (e.g., schedules, IP addresses, routes). The *Network Manager* keeps track of network topology changes such as link speeds between nodes, addition or removal of nodes, and handovers on Wi-Fi domain.

Figure 3 shows the message flow between CNC and TSNA at different stages of network operation. On startup, a configuration file is loaded with basic settings for nodes and the TSNC itself, where each NE is identified by a Unique Identifier (UID) (e.g., based on hostname, or MAC addresses). The CNC opens a socket for TSNAs to connect. A TSNA connects announcing its resources and its UID, and if there is an entry for that UID in the configuration file, the specific configuration for the NE is transmitted back to the TSNA, otherwise, a generic configuration is used. New configurations can be transmitted to the TSNAs at any moment. Relevant changes on node state

[3]https://zeromq.org/

(e.g., interface state, data rate) are reported immediately by TSNAs to the CNC.

The CNC keeps a hash table with one entry per NE, holding relevant information such as interface states, link data rates, number of transmission/reception queues, state of processes for PTP synchronization, and INT. The hash table stores the latest information only, in order to avoid excessive memory usage, while persistent and long-term records (e.g., change of interface data rate over time) are recorded in the database of the *Monitor* module. A node entry contains a *Communication Interface* object that abstracts the communication protocol to the specific node. Thus, configuration changes and information-gathering commands are properly translated by this interface.

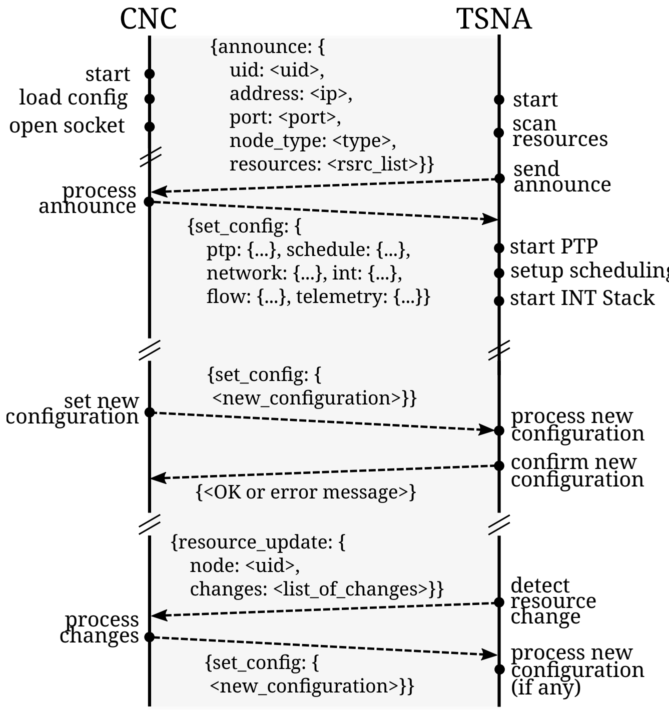

CNC
TSNA
start
load config
open socket
start
scan resources
send announce
{announce: { uid: <uid>, address: <ip>, port: <port>, node_type: <type>, resources: <rsrc_list>}}
process announce
{set_config: { ptp: {...}, schedule: {...}, network: {...}, int: {...}, flow: {...}, telemetry: {...}}
start PTP
setup scheduling
start INT Stack
set new configuration
{set_config: { <new_configuration>}}
process new configuration
confirm new configuration
{<OK or error message>}
{resource_update: { node: <uid>, changes: <list_of_changes>}}
detect resource change
process changes
{set_config: { <new_configuration>}}
process new configuration (if any)

Fig. 3: Communication flow between TSNC and TSNA

Figure 4 illustrates the abstraction of communication methods provided by the *Communication Interface*. The TSNA of a *Node X* announces itself using a specific *Driver*. After processing the announce, the TSNC adds Node X to its hash table and the *CommIface* object utilizes the corresponding driver (*Driver 2*). When the configuration commands are issued to the driver (e.g., set traffic filtering rules, set schedule), the driver translates them internally to the method being used by the node, such as NETCONF using YANG models.

C. Management Interface

The *Management Interface* is split into two components. The UNI provides functions for Talkers and Listeners to inform their network requirements to the controller. It also offers functions for network operators to manage the TSNC elements and the TSN network as a whole, being the single point of access for user operation. Therefore, the UNI is also responsible for user authentication and permission control.

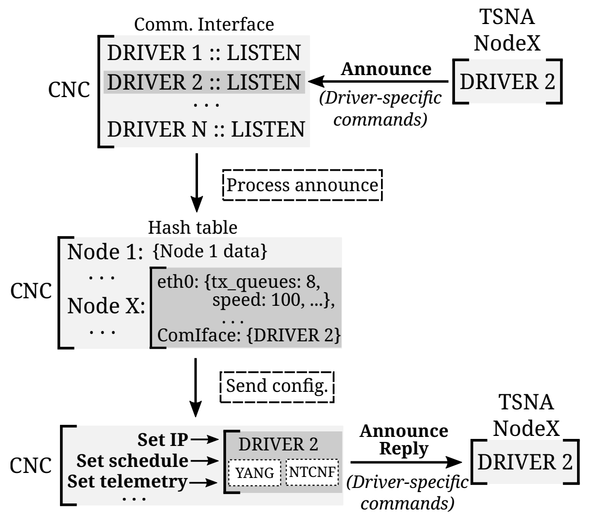


Fig. 4: Driver management by the Communication Interface

Inter-module communication is done through the *Internal Interface*. It validates the commands to be sent to the network and provides a central point of intercommunication between modules (i.e., CNC, CUC, Scheduler, Control Loop, Monitor). Through this interface, the modules can request information such as topology, link states, and installed schedules, and apply configurations to NEs based on their UIDs. The Management Interface is effectively part of the CNC, however, the coupling between CNC and Internal Interface is higher than between UNI and CNC.

D. Monitor

Monitoring is an essential task in most networks, especially those supporting critical activities like TSNs. Information such as link states is directly received by the CNC through notification from TSNA and updated on the Database. On the other hand, monitoring performed using INT such as delays (hop-by-hop and end-to-end), Packet Loss Rate (PLR), throughput, and others, are managed by an independent *Monitor* module. The *Monitor* module is composed of a *Subscriber* that receives monitoring data from NEs through publish/subscribe mechanism. A *Dashboard* submodule is responsible for data filtering, display, and setting alarms.

Despite already having communication sockets between TSNAs and the CNC, through which network telemetry could be transmitted, we define the socket to the monitor as an independent socket to give the flexibility to deploy the Monitor on a different node. This also allows monitoring to continue as an independent service, and trigger alarms to the operator even in face of a catastrophic failure of the CNC.

E. Centralized User Configuration – CUC

The CUC is responsible for high-level network coordination and registration of applications with strict performance requirements executed in the TSN network. Applications are registered with the specification of their traffic patterns, a 5-tuple for traffic identification, and the required network performance between source and destination. This information is used by the Scheduler to define routes and traffic shaping rules to be applied on NEs, and by the Control Loop to monitor and take control decisions in the network. The items of the application specification may vary according to the use case. The list presented here is sufficient for the use cases in the projects we are involved in, however, it is not exhaustive. An application is registered with the following elements:

- **Application ID:** the unique identifier of the application in the network.
- **Flow identification 5-tuple:** Source IP, Destination IP, Source Port, Destination Port, Protocol. IPs are IPv6 addresses, and Protocol refers to transport layer protocols.
- **Application specification:** each item is specified for downlink (source to destination) and uplink (destination to source).
  - Transmission cycle (duration of a packet burst in ms)
  - Frames per cycle
  - Frame size (Bytes)
- **Application requirements:** each item is specified for downlink (source to destination) and uplink (destination to source). In addition, each value is specified as a tuple of minimum and maximum tolerated values. For example, the delay of an application can be specified as $[10, 12]$ to indicate that up to 12 ms is tolerable, or $[10, 10]$ to specify that the network delay must be exactly 10 ms.
  - Throughput (bits per second)
  - Delay (ms)
  - Jitter (ms)
  - Loss (percentage)

From the application registry, the *Network Orchestrator* derives monitoring rules for In-band Telemetry and creates a map between application ID and telemetry rules, in order to simplify the process of matching application requirements and the delivered network performance. The orchestrator also interacts with the Scheduler and Control Loop to define a configuration that complies with the requirements of all the registered applications and defines rules for the continuous monitoring of traffic flows. By means of the CUC we implement the Fully Centralized configuration model specified on IEEE 802.1Qcc [19].

F. Scheduler

Defining a schedule for a TSN is a complex task, in fact, it is equivalent to the bin-packing problem in the case of time-based schedules, known to be NP-complete [43]. In this sense, generating TSN schedules can demand significant computational power, especially for networks with a large number of nodes or flows. In our architecture, we specify the Scheduler as an independent pluggable module as the scheduling requirements on TSN networks vary significantly according to the use cases. Small and static networks can benefit from schedulers that employ exact solvers, as schedule changes occur infrequently or might never occur. On the other hand, networks with mobile NEs (e.g., with Wi-Fi devices) might require frequent schedule changes, and can benefit from faster but less accurate methods using AI or genetic algorithms. Moreover, the Scheduler module can be offloaded to more capable nodes, to avoid increasing the resource requirements of the TSNC.

The information required by TSN schedulers existing in the literature varies according to their objectives and complexity. Below, we list the inputs required by three different TSN schedulers [44]–[46] and which component of our architecture provides such information:

- **Network topology:** provided by the CNC, either based on information from the network configuration file or by using Link Layer Discovery Protocol (LLDP). The **number of queues** and **link data rate** of each interface are also provided.
- **Flows and application requirements:** provided by the CUC, based on the registered applications.
- **Per-hop delay (comprising processing and propagation):** provided through real-time in-band monitoring of switch and router performance. This information depends on telemetry services and can be retrieved by consulting the database.

The output of the scheduler must be a set of parameters for the configuration of the desired traffic scheduling mechanism, along with routing instructions for a flow. This output can be either a Gate Control List (GCL) compliant with IEEE 802.1Qbv [15] or a set of parameters (i.e., per-queue idle slope, send slope, high credit, low credit) compliant with IEEE 802.1Qav [14]. For Wi-Fi nodes, the schedules must include silent periods (when no queues are scheduled), taking into consideration the shared nature of the medium, leaving time for other nodes to access the spectrum. The scheduler module must provide an interface for interaction with other nodes with at least two capabilities: i) *get current schedule:* which returns the schedule currently in force; ii) *generate new schedule:* which calculates and returns a new schedule with a unique identifier. This schedule can be retrieved by other modules (such as the Control Loop) and validated before being applied to the network.

G. Control Loop

The *Control Loop* is a module implementing automated real-time control in the TSN. This module is in charge of assessing the compliance of the network with the performance requirements registered for the applications in the CUC. To do so, the control loop consults the telemetry reports in the monitor database and compares the measured performance against the KPIs specified for an application. Further information that might be necessary, such as network topology and the state of critical processes can be retrieved from the CNC.

Based on the gathered information, the control loop employs algorithmic or AI-based control methods to generate new rules (e.g., defining new routes, requesting new traffic shaping rules to the Scheduler) and apply them in the network through the CNC. While the scheduler module is also in charge of generating routing/traffic shaping rules, the action of triggering network reconfiguration during steady-state operation based on real-time telemetry information is assigned to the control loop module.

V. TSN Agent

The TSN Agent (TSNA) is what enables the TSNC to perform fine-grained control over NEs, interacting with the CNC through the *Communication Interface* and a specific *Driver*. The *Communication Interface* of the TSNA has a similar operation to its counterpart on the CNC, i.e., to select the appropriate driver to communicate with the CNC. Figure 5 shows the organization diagram of the TSNA.

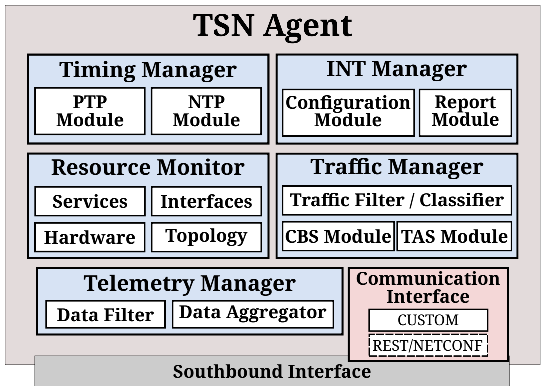


Fig. 5: TSN Agent submodules

The TSNA is initialized specifying the UID and type of the NE. On startup, the TSNA scans the NE capabilities (e.g., network interfaces, number of transmit and receive queues, link data rates, neighbor nodes), and announces itself to the TSNC. All the NE information is transmitted in the announce message. The CNC replies with the necessary configurations to initialize the TSNA modules. Figure 3 illustrates the initialization process of TSNA and message flow with the CNC. The reply message from the CNC contains the following information:

- **PTP configuration:** determines the role of the NE for time synchronization (e.g., GrandMaster (GM), Slave, Boundary Clock (BC)), and the interfaces to be used.
- **Schedule:** gate control list for interfaces of the NE (example shown on Figure 6) and the base time for the schedule.
- **Network configuration:** defines IPv6 addresses for the data plane, routes, and interfaces to be bridged.
- **INT configuration:** determines the interfaces that will be capable of performing in-band telemetry. For more information about the INT framework used in this work we refer the reader to the work of Haxhibeqiri et al. [40].
- **Flow configuration:** defines rules to apply filtering and policing for specific flows.
- **Telemetry:** defines what information should be reported to the CNC, as well as rules for telemetry data filtering and aggregation to reduce monitoring overhead.

A. Timing Manager

The *Timing Manager* controls the time synchronization services on a NE. The Network Time Protocol (NTP) module synchronizes the clock of the NE defined as PTP GM with a global reference of time. Then, the PTP module synchronizes the NEs of the TSN using the Precise Time Protocol. The PTP module is based on *linuxptp*[4] and is responsible for configuring and running the *ptp4l* and *phc2sys* processes. The module forwards reports synchronization state and accuracy reports to the Telemetry Manager.

B. INT Manager

The INT manager builds on top of the In-band telemetry framework presented by Haxhibeqiri et al. [40], implemented using Click Router [47]. The framework adds telemetry information into data packets to provide per-flow/per-hop measurements of throughput, delay, jitter, and packet loss ratio, allowing fine-grained monitoring of the performance of network flows. INT is configured through the Configuration Module, based on the application registration data stored in the CUC. The configuration contains the Application ID, the Flow Identification 5-tuple, and how often the telemetry information should be added to packets (i.e., insert telemetry headers every N packets to avoid excessive telemetry overhead).

The INT framework periodically generates reports about flow performance, to be published to the TSNC (Monitor). The Report Module performs a pre-processing of the raw report that contains only counter values (e.g., packets transmitted/received) and timestamps (i.e., the time at which a packet traversed each node), and calculates Quality of Service (QoS) information in terms of throughput, delay, jitter, and packet loss ratio. The resulting report is sent to the Telemetry Manager that can further filter and aggregate the reports before transmitting them to the controller.

C. Resource Monitor

The resource monitor is responsible for constantly monitoring the state of the NE in terms of services, network interfaces, hardware, and topology changes. During the startup of the TSNA, the resource monitor scans the node capabilities and transmits them in the announce message. On steady-state operation, the resource monitor keeps monitoring the services/devices/topology, transmitting only the updates to the controller.

In terms of *services*, the states of the following processes are monitored:

- **ptp4l**: responsible for PTP synchronization.
- **phc2sys**: for synchronizing system and network clocks.
- **Click**: for INT and time-based scheduling on top of openwifi.
- **lldpd**: for neighbor discovery and topology monitoring.

Regarding network interfaces, the resource monitor keeps track of their state (up or down) and link speed. This works in combination with the *topology* monitoring, detecting when links between NEs change in terms of connection speed, or when a link is broken. In the Wi-Fi domain, the topology information includes data about associated clients. Finally, the *hardware* monitoring comprises temperatures, voltages, and other information that are reported when reaching critical values.

[4]http://linuxptp.sourceforge.net/

D. Traffic Manager

Functions related to traffic classification, filtering, and shaping are carried out by the *Traffic Manager*. This module then defines how to apply them in the interfaces according to interface type and capabilities. On wired interfaces, the schedules are applied using the Traffic Control (TC) (tc-taprio[5]) tool. For Wi-Fi interfaces, the schedule specification is first converted to the format used in openwifi, and then applied to the interface. This operation is transparent to the controller – a benefit of the Controller/Agent architecture, as the TSNAs abstract the specificity of different devices and provide a homogeneous configuration interface.

The example in Figure 6 shows the configuration received by a node with UID *ap1*, with a schedule to be applied on the wired interface *eth1* and on the Wi-Fi interface *sdr0*. Schedules based on IEEE 802.1Qbv are received from the controller in a standard format and defined as a GCL. The GCL specifies the gates to be open at a given moment and the duration of a slot in nanoseconds. The sum of times of all the GCL entries is the cycle duration of the schedule. The GCL for Wi-Fi interfaces must contain intervals where all gates are closed (highlighted in bold in the figure), which determines that no queue is served during these time slots. Due to the shared nature of the wireless medium, these intervals are used either by a client to transmit to the Access Point (AP) or as spare periods that can be allocated to other flows.

Time-aware scheduling based on IEEE 802.1Qbv is controlled by the Time-Aware Scheduling (TAS) Module, while the shaping mechanism based on IEEE 802.1Qav is controlled by the Credit-Based Shaper (CBS) Module.

```
"ap1": {
  "schedule": [
    {"interface": "eth1",
     "gcl": [{"gate_mask": "0x80", "duration": 100000},
             {"gate_mask": "0x40", "duration": 200000},
             {"gate_mask": "0x3f", "duration": 700000}],
     "base-time": <timestamp>
    },
    {"interface": "sdr0",
     "gcl": [{"gate_mask": "0x00", "duration": 8000000},
             {"gate_mask": "0x01", "duration": 8000000},
             {"gate_mask": "0x00", "duration": 8000000},
             {"gate_mask": "0x02", "duration": 8000000},
             {"gate_mask": "0x00", "duration": 8000000},
             {"gate_mask": "0x03", "duration": 8000000},
             {"gate_mask": "0x00", "duration": 8000000},
             {"gate_mask": "0x04", "duration": 8000000}],
     "base-time": <timestamp>
     }
  ]
}
```

Fig. 6: Example of schedule configuration for an Access Point

E. Telemetry Manager

The *Telemetry Manager* controls what information is reported to the CNC, allowing telemetry reports to be filtered, aggregated, or disabled. The data filter can be configured to allow the transmission of telemetry reports only when the values exceed predefined thresholds. Data can also be aggregated and transmitted as statistics over a time period (e.g., 99th percentile of synchronization error in the last minute). This can contribute to reducing the monitoring overhead in the network but still allows telemetry reporting to be dynamically adjusted for troubleshooting.

VI. Experimental Setup

In order to report on the key performance indicators of our implementation, we have carried out three sets of experiments. The first set analyzes the performance of the controller for networks with different numbers of nodes. The objective is to understand the resource usage in terms of CPU, memory, and network, giving a clear indication of the hardware requirements to deploy the controller and its scalability. We use a controlled wired setup with five nodes on the Virtual Wall[6] testbed: one operated as the TSNC (running the TSNC), and the other four executed docker containers running the TSNAs to emulate a network with a large number of NEs. The four nodes shared the same link to the TSNC, allowing us to monitor the network load generated by the TSNAs.

In the second and third sets of tests, we validate the architecture by performing monitoring and control activities through in-band telemetry, precise time synchronization, and dynamic cross-domain scheduling. The second set of experiments uses the topology from Figure 7, consisting of eight nodes on the Virtual Wall testbed: four wired end nodes, three switches, and one controller. The dashed lines represent logical connections between the nodes and the controller. The dotted lines show the traffic generated during our validation experiments. Two UDP applications generated packets every 1 millisecond from PC1 to PC2, while from PC3 and PC4 we used iperf3[7] to generate flooding traffic between PC3 and PC4. During the experiments, we apply different scheduling and traffic classification rules on PC1, SW1, SW2, and SW3 to give each flow different end-to-end performance prioritization.

In the third set of experiments we used the topology on Figure 8, deployed on w-iLab.t[8], a large-scale open Fed4FIRE+ [48] testbed intended for Wi-Fi and sensor networking experimentation. This setup has one wired node, one switch, one Wi-Fi AP, and two Wi-Fi clients based on openwifi. We generated UDP traffic from both clients to the PC1, applying schedules with openwifi on both clients, the AP, and SW1.

All nodes used on Virtual Wall consist of Intel(R) Xeon(TM) CPU E5645, with 24GB of RAM, equipped with Intel 82576 Gigabit Ethernet controllers. For each port, the controller offers 8 transmission and 8 reception hardware queues. The operating system used was Ubuntu 20.04 LTS with kernel 5.15.0. On w-iLab.t testbed the AP consisted of a Zynq UltraScale+ MPSoC ZCU102[9] board and

[5]https://man7.org/linux/man-pages/man8/tc-taprio.8.html

[6]https://doc.ilabt.imec.be/ilabt/virtualwall/

[7]https://iperf.fr/

[8]http://doc.ilabt.imec.be/

[9]https://www.xilinx.com/products/boards-and-kits/ek-u1-zcu102-g.html

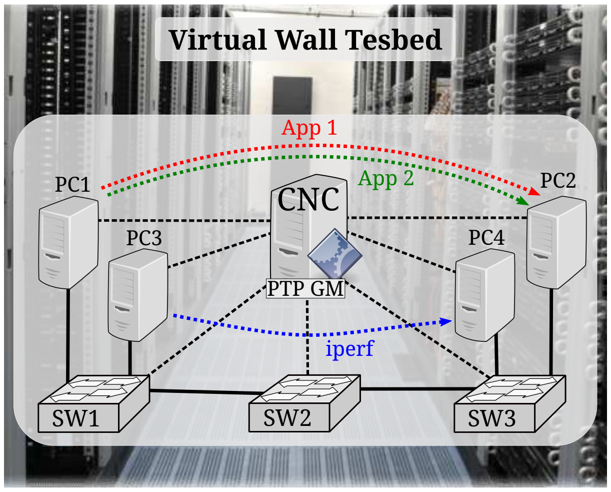


Fig. 7: Wired testbed setup on Virtual Wall

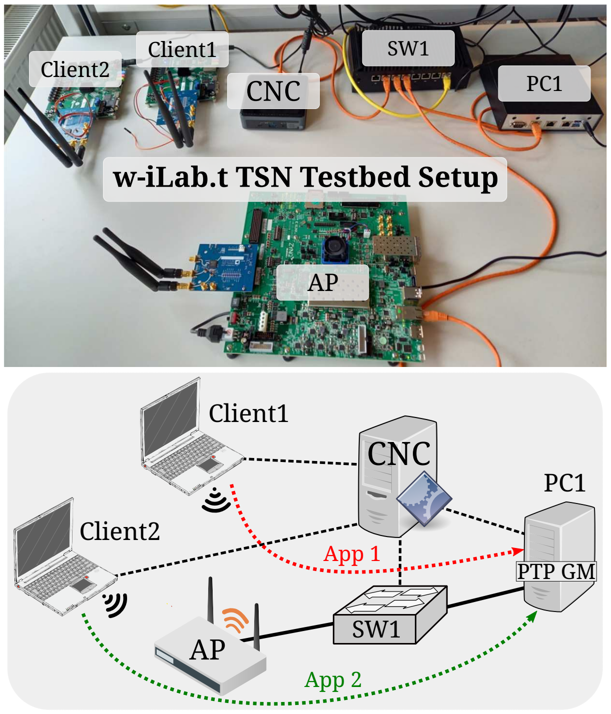


Fig. 8: Wi-Fi testbed setup w-iLab.t

AD-FMCOMMS2-EBZ[10] radio frontend to communicate to Clients 1 and 2. The clients were running on Zynq-7000 SoC ZC706[11] boards. The CNC and PC1 on w-iLab.t testbed are Intel NUC nodes with Intel(R) Core(TM) i7 CPU and Intel I219-V Ethernet controllers. The switch is an Evrtech KBOX-3102 industrial switch with an Intel(R) Core(TM) i7 CPU and Intel I211-AT Ethernet controller.

The CNC modules were created in Python 3 programming language. We used the PostgreSQL[12] as database, and Grafana[13] for the dashboard in the Monitor. The TSNA is also based on Python 3. In the wired setup (Figure 7) we configured the TSNC as GM, and the switches as BCs providing synchronization to the PCs, which operated as PTP Slaves. In the hybrid setup (Figure 8), PC1 was a GM, SW1 was configured as a BC, and the AP using openwifi uses a particular configuration: operates as a Slave on the wired interface, and as a GM on Wi-Fi interface. PTP Hardware Clocks (PHCs) of wired and wireless interfaces are synchronized using *phc2sys* (from *linuxptp* package) to allow global time synchronization of both LAN and WLAN domains. The two Wi-Fi clients operate as PTP Slaves.

VII. Results

In this section, we present the results of the experiments showing the feasibility of our architecture. First, we show results related to the performance of the TSNC (specifically, the CNC module) when handling simultaneous requests from many TSNAs, as well as CPU, Memory, and Network usage footprints. These experiments are carried out using the TSNAs running on docker containers. Then, we validate the proposed controller by configuring and managing LAN and WLAN TSN domains on Virtual Wall and w-iLab.t.

A. *TSN Controller Performance*

We first analyze the agent serving time when increasing numbers of TSNAs try to connect to the CNC at the same time. Each TSNA was executed in a docker container with a varying number of virtual interfaces, and announcing themselves as different NE types (i.e., router, switch wired node, AP, Wi-Fi node). To avoid processing bottlenecks on the TSNA side we executed up to 250 containers on each Virtual Wall node. We measured the time difference between a TSNA announcing itself, and receiving the announce reply from the CNC.

Results shown in Figure 9 indicate that the CNC was able to process 200 simultaneous requests with a median time below one second. For 10 simultaneous requests, the median was 25 milliseconds. In the case with 1000 simultaneous requests, the median time was 6 seconds, with the longest serving time observed at 11.8 seconds. The CNC showed sufficient performance on agent serving time even for hundreds of simultaneous requests. Considering that PTP synchronization can take several seconds to converge to sub-microsecond accuracy, the agent serving time of our CNC does not represent a bottleneck for the TSN network bootstrap.

Figure 10 shows the memory footprint for an increasing number of associated NEs. As the CNC keeps the internal hash table with the information on NEs, it is expected that the memory consumption increases as more NEs connect. Each bar in the graph shows the base memory (used by the TSNC before any TSNA associates) and the final memory usage after all the TSNAs associate. For each run we restarted the TSNC, therefore, minor oscillations of the base memory usage were observed. Also, in all runs, the configuration file contained entries for all the 1000 NEs that could connect to the CNC, which also increases the base memory usage. In the scenario with 1000 NEs, the total memory usage was 121 MB, which represents approximately 123 KB of memory for each TSNA.

[10]https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-ad-fmcomms2.html
[11]https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html
[12]https://www.postgresql.org/
[13]https://grafana.com/

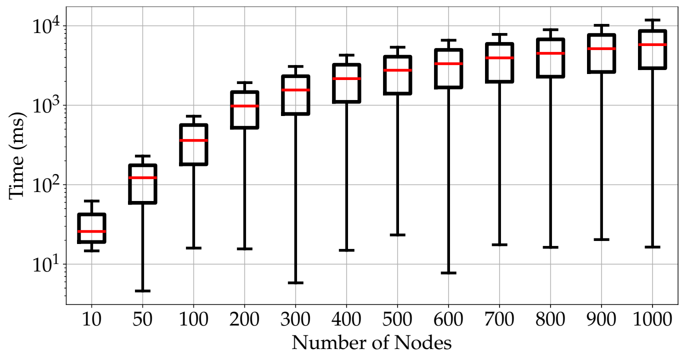


Fig. 9: Agent serving time

These results show the low memory footprint of the CNC per associated NE, leaving such resources to be allocated to more complex modules like Scheduler and Control Loop.

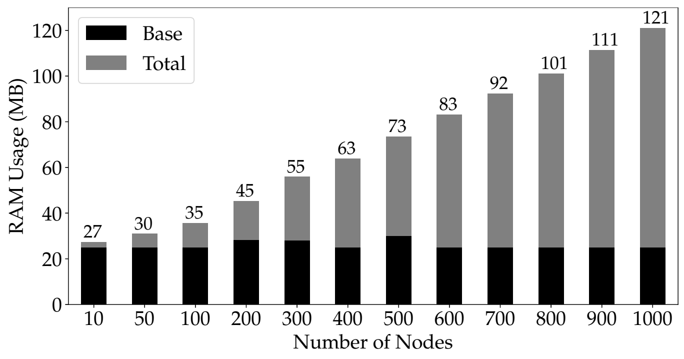


Fig. 10: TSNC memory usage per number of agents

In Figure 11 we show the CPU usage by the CNC while processing the simultaneous association requests. We select the results for 50, 100, 500, and 1000 NEs as they represent cases with small (50 and 100), average (500), and high (1000) amounts of requests. For 10 simultaneous requests, there was no significant load. For 50 and 100 we observed a quick increase to approximately 18 % of CPU usage. For 500 and 1000 requests the initial CPU usage is approximately 37 %, followed by 5 and 10 seconds respectively of approximately 18 % load. CPU usage results show that the TSNC does not need to run on powerful servers and that similarly to memory, CPU resources can be allocated to more complex modules.

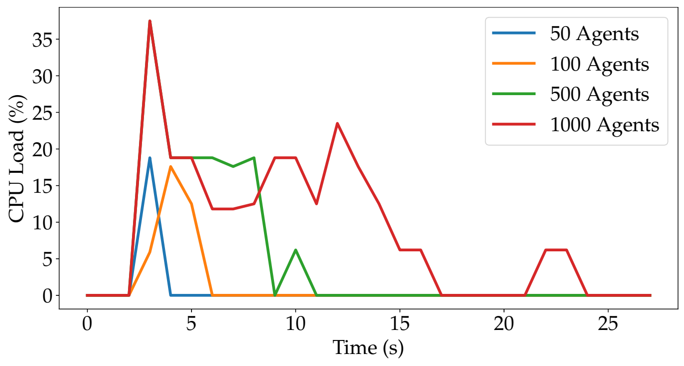


Fig. 11: TSNC CPU usage during association of agents

Lastly, we show in Figure 12 the network input and output traffic generated during association requests. Solid lines show the traffic load for 50 agents, while dashed lines show the traffic load for 1000 agents. Input traffic is higher at startup when nodes transmit their capabilities during announce. However, the output network load is higher and spans for a longer period as the CNC transmits back the configuration for each NE, including configuration files (e.g., for the INT framework).

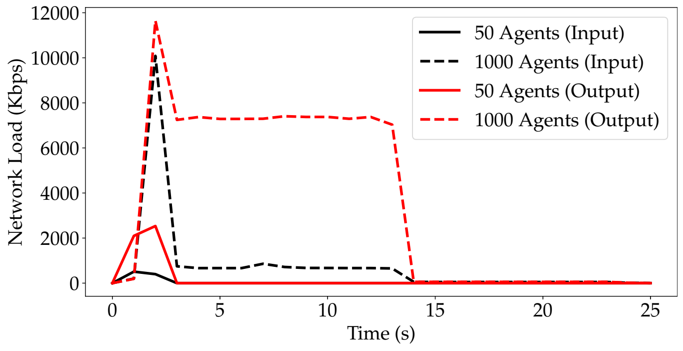


Fig. 12: TSNC network traffic during association of agents

The performance evaluation shows that the CNC is capable of handling many simultaneous requests with low CPU, memory, and network footprints, which allows resources to be allocated to other more demanding services. However, the agent serving time increases as more simultaneous requests occur. Currently the CNC processes requests in a First In, First Out (FIFO) scheme, however, the low CPU and memory requirements indicate that further optimizations can be implemented, such as performing parallel processing of requests.

B. *Testbed Validation*

We now show the validation results by building two different TSN networks on distinct testbeds and managing TSN services centrally from the controller. First, we show the synchronization accuracy achieved on each testbed, as a common and accurate reference of time is crucial for the operation of a TSN network.

1) Synchronization: Synchronization results are described in Table II. The values were obtained from *linuxptp*, which returns estimates of clock offset from the GM (NEs operating as GM are omitted on the table). The first column describes the NE, the second column shows the median absolute offset, and the third column shows the 90th percentile of absolute offset. In Virtual Wall we observed lower offsets on the switches, with medians between 213 and 255. The offsets on PCs were significantly higher and with higher oscillation (ranging from $3\mu s$ to 15 μs in the median, with peaks of over 1 ms). This variation is expected due to testbed setups: we do not have control over which nodes of the testbed were allocated for the experiment, and the testbed infrastructure contains switches between the nodes that we cannot control. Thus, synchronization errors can be introduced by the switches. Also, the PCs are one hop farther from the GM and get synchronization from the BCs. Therefore, higher offsets are expected from the PCs.

TABLE II: PTP synchronization offset from GM (in nanoseconds) on testbeds

Node	Median	90th Percentile
Virtual Wall		
PC1	3055	9257
PC2	6835	97281
PC3	14965	49414
PC4	3535	8306
SW1	213	555
SW2	255	587
SW3	225	580
w-iLab.t		
SW1	211	529
AP	939	1955
Client 1	1000	2500
Client 2	500	2500

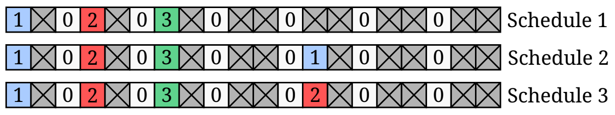
In the Wi-Fi testbed the synchronization offset between SW1 and GM had a median absolute offset of 211 ns, with a 90th percentile of 529 ns. The wired interface of the AP showed a median offset of 939 ns and a 90th percentile of 1955 ns. Synchronization errors between AP and clients were in line with the results presented by Aslam et al. [13], with a median of 1000 ns and 500 ns for Clients 1 and 2 respectively, with a 90th percentile of 2500 ns.

2) Scheduling: In scheduling experiments, we demonstrate fine-grained control over traffic by using the CNC to manage the TAS parameters on NEs. We use two applications to measure network performance and the effects of the applied schedules. We use *iperf* to generate TCP traffic, and a custom UDP traffic generator, in which a client tags packets with the current timestamp and transmits them to the server in predefined intervals. The server receives the packets, calculates the one-way delay, generates statistics every second, and publishes them to the Monitor module. As the NEs are PTP synchronized, the one-way delay error is within synchronization accuracy. We configured the UDP traffic generators (i.e., *App 1* and *App 2*) to send packets every 1 ms.

Figure 13 shows the three schedules used for experiments on the wired testbed. Each square represents a slot of 250 μs, resulting in a cycle of 5 ms. Crossed squares are slots in which all the gates are closed (there is no transmission during those slots). Such slots can be allocated, for example, to new flows being started on the network. PTP and best-effort traffic are transmitted on queue 0, therefore, we allocate 30 % of the cycle duration to queue 0, distributed during the cycle.

We assign *iperf* traffic to queue 1, *App 1* to queue 2, and *App 2* to queue 3. Considering that all interfaces run at 1 Gbps, the expected theoretical throughput achieved by *iperf* on Schedules 1 and 3 is 6.25 MB/s, as 1/20th of the 5 ms cycle is reserved for queue 1 on both schedules. Schedule 2 adds another slot to queue 1, doubling the throughput. Apps 1 and 2 start with the same priority, and on Schedule 3 we add one more slot to App 1, giving it more transmission opportunities in a cycle and reducing the end-to-end delay.

The results are reported in Figure 14, showing when each schedule is in force. The one-way delay represents the 99th percentile for packets received during the last second by the


1 ⊠ 0 2 ⊠ 0 3 ⊠ 0 ⊠⊠ 0 ⊠⊠ 0 ⊠⊠ 0 ⊠⊠ Schedule 1
1 ⊠ 0 2 ⊠ 0 3 ⊠ 0 ⊠⊠ 0 1 ⊠ 0 ⊠⊠ 0 ⊠⊠ Schedule 2
1 ⊠ 0 2 ⊠ 0 3 ⊠ 0 ⊠⊠ 0 2 ⊠ 0 ⊠⊠ 0 ⊠⊠ Schedule 3
Fig. 13: Schedules for wired testbed experiments

UDP App server. Delay oscillations were caused by peaks of synchronization error observed on PC2 during experiments. The performance indicators were in line with the expected behavior from the schedules. With a cycle duration of 5 ms, and one slot per cycle allocated to Apps 1 and 2, most of the packets being generated were buffered during the 19 slots (4.75 ms) while their respective queues were not scheduled for transmission. When one more slot is added to App 1 (schedule 2), the buffering time is at most 2.75 ms, in line with the observed results. App 2 kept the same behavior during the experiment, not being affected by schedule changes for other flows, as its slot allocation was kept the same all the time. The effective throughput achieved with *iperf* was also in line with expectations, taking into consideration that the real throughput is slightly lower than the theoretical and that other switches exist between the nodes. Applying Schedule 2 we double *iperf* throughput, but do not affect the one-way delay of the UDP Apps. After applying Schedule 3, *iperf* returns to the initial behavior, while App 1 gets higher priority than App 2, having its packets delivered with a lower delay.

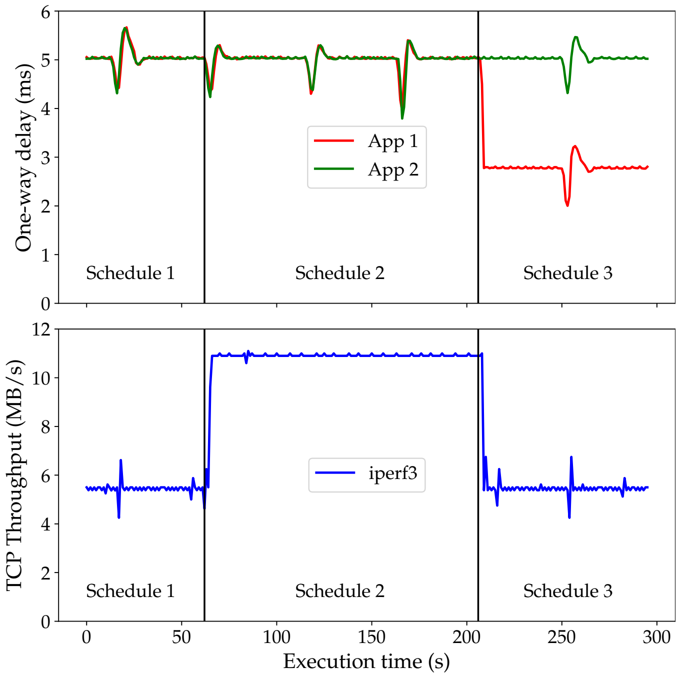


Fig. 14: Effect of schedules on traffic performance on the wired testbed with the topology from Figure 7

In the Wi-Fi testbed, we used the Schedules shown in Figure 15 to control uplink traffic from Clients 1 and 2 to PC1. The Wi-Fi schedules are configured taking into account the shared nature of the wireless medium, with silent periods to avoid collisions between flows coming from different nodes, and at the end of the cycle to allow association of new nodes [49].

In Figure 15 we describe the schedules for each device, considering the path of the flows from the Wi-Fi clients to the wired node. Each square represents a slot of 256 μs, resulting in a cycle of 5.12 ms. We used two UDP App instances to verify the effects of scheduling. App 1 was allocated to queue 1 and App2 was allocated to queue 2. Schedule 1 gives the same number of slots to both queues. On Schedule 2 we add a 1 ms slot to queue 1 on SW1, while the AP and Wi-Fi clients have a 2 ms slot at the end of the cycle allocated to queues 0, 1, and 3. Therefore, Schedule 2 gives more priority to App 1 than to App 2. The logic is inverted on Schedule 3, giving more priority to App 2. App 1 traffic is generated at client 1 while the App 2 traffic flow is generated at Client 2 both with a destination at PC1.

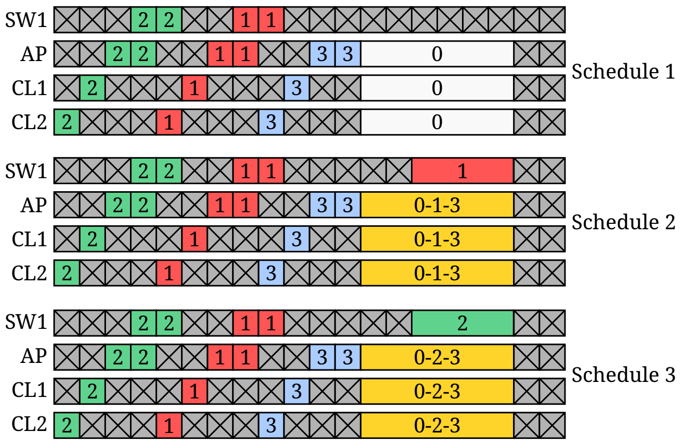


Fig. 15: Schedules for Wi-Fi testbed experiments

Results for Wi-Fi scheduling are shown in Figure 16. Using Schedule 1 (at the beginning and the end of the experiment), both flows show the same delay pattern. Schedule 2 gives more priority to App 1 with the additional slot, reducing the delay from 5 ms to 3 ms. Schedule 3 gives more priority to App 2 on Client 2, albeit with a larger variation. The variation is caused by the number of slots between the two transmission chances given to queue 1 and queue 2 on the Wi-Fi clients. While queue 1 is not scheduled for at most 6 slots (on Schedule 2, CL1), queue 2 is not scheduled for 11 slots (on Schedule 2, CL2), forcing packets of App 2 to be buffered for longer periods. Nevertheless, the results show that traffic scheduling can be correctly performed and controlled by the CNC through the specification of GCLs for wired and Wi-Fi devices.

VIII. Conclusion and Future Work

In this work, we describe and implement a modular architecture for a multi-domain end-to-end TSN CNC. We identify the technical details of the main modules required by the TSN CNC, as well as by the agent that manages the NEs, in order to efficiently support the tasks required by a TSN network comprising wired and Wi-Fi NEs. We implement the main modules, namely TSN Controller, Communication Interface, a Custom Driver for communication, and the TSN Agent that runs on NEs. By defining a Communication Interface that abstracts the communication method used by

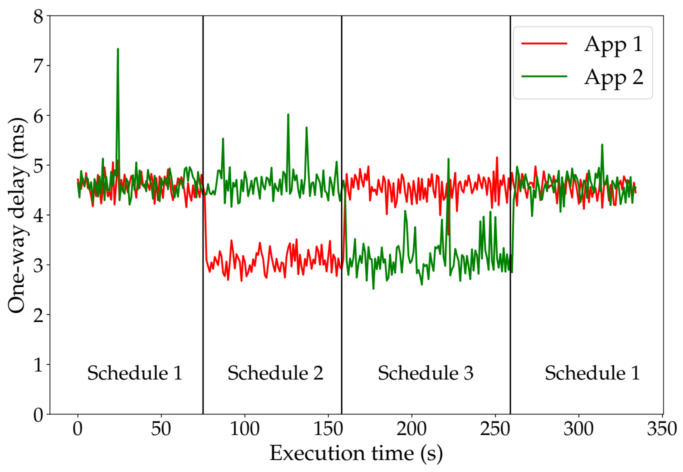


Fig. 16: Effect of schedules on traffic performance on Wi-Fi testbed with the topology from Figure 8

the TSN network devices, the CNC can support devices that employ different mechanisms for device configuration, either a proprietary driver or open standards like NETCONF. Our architecture provides support for PTP synchronization, bounded low latency, and resource management in an end-to-end manner for both Wi-Fi and wired domains. We discuss on the technical details and existing challenges of our proof-of-concept implementation, evaluating its performance in terms of scalability and demonstrating its end-to-end feasibility by running experiments on two different testbeds. We set up wired and Wi-Fi TSNs nodes in a joint manner, and demonstrate the end-to-end precise time synchronization and fine-grained traffic shaping achieved by the full setup.

Our work pushes further the existing research on TSN by presenting a flexible and scalable platform that may serve as the scaffolding for new end-to-end TSN solutions. Future work should aim to leverage the flexibility of this CNC architecture to fully automate, tentatively in an intelligent manner, the TSN control on top of the existing functionalities to offer a continuum TSN management in heterogeneous wired/Wi-Fi networks.

Acknowledgment

This research is partially funded by the imec ICON project VELOCe - VErifiable, LOw-latency audio Communication (Agentschap Innoveren en Ondernemen project nr. HBC.2021.0657).

References

[1] N. Finn, "Introduction to Time-Sensitive Networking," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 22–28, 2018. [Online]. Available: doi.org/10.1109/MCOMSTD.2018.1700076
[2] J. L. Messenger, "Time-Sensitive Networking: An Introduction," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 29–33, jun 2018. [Online]. Available: doi.org/10.1109/mcomstd.2018.1700047
[3] "Time-Sensitive Networking: A Technical Introduction," *Cisco Public White Paper*, 2017. [Online]. Available: www.cisco.com/c/dam/en/us/solutions/collateral/industry-solutions/white-paper-c11-738950.pdf
[4] D. Cavalcanti, "Wireless TSN – Definitions , Use Cases & Standards Roadmap," *Avnu Alliance*, pp. 1–16, 2020. [Online]. Available: avnu.org/wireless-tsn-paper/

[5] Z. Li, M. A. Uusitalo, H. Shariatmadari, and B. Singh, "5G URLLC: Design Challenges and System Concepts," in *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*, 2018, pp. 1–6. [Online]. Available: doi.org/10.1109/ISWCS.2018.8491078

[6] T. Adame, M. Carrascosa, and B. Bellalta, "Time-Sensitive Networking in IEEE 802.11be: On the Way to Low-latency WiFi 7," dec 2019. [Online]. Available: http://arxiv.org/abs/1912.06086

[7] X. Jiao, W. Liu, M. Mehari, M. Aslam, and I. Moerman, "openwifi: a free and open-source IEEE802.11 SDR implementation on SoC," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, 2020, pp. 1–2. [Online]. Available: doi.org/10.1109/VTC2020-Spring48590.2020.9128614

[8] J. Haxhibeqiri, X. Jiao, M. Aslam, I. Moerman, and J. Hoebeke, "Enabling TSN over IEEE 802.11: Low-overhead time synchronization for Wi-Fi clients," in *ICIT2021, the 22nd International Conference on Industrial Technology*, 2021, pp. 1068–1073. [Online]. Available: doi.org/10.1109/ICIT46573.2021.9453686

[9] L. Lo Bello and W. Steiner, "A Perspective on IEEE Time-Sensitive Networking for Industrial Communication and Automation Systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1094–1120, 2019. [Online]. Available: doi.org/10.1109/JPROC.2019.2905334

[10] S. Waldhauser, B. Jaeger, and M. Helm, "Time Synchronization in Time-Sensitive Networking," no. April, pp. 2–7, 2020. [Online]. Available: doi.org/10.2313/NET-2020-04-1

[11] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications," *IEEE Std 802.1AS-2020*, pp. 1–421, 2020. [Online]. Available: doi.org/10.1109/IEEESTD.2020.9121845

[12] M. Aslam, W. Liu, X. Jiao, J. Haxhibeqiri, J. Hoebeke, I. Moerman, E. Municio, G. Miranda, P. H. Isolani, and J. M. Marquez-Barja, "High Precision Time Synchronization on Wi-Fi based Multi-Hop Network," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2021, pp. 1–2. [Online]. Available: biblio.ugent.be/publication/8709058/file/8709060.pdf

[13] M. Aslam, W. Liu, X. Jiao, J. Haxhibeqiri, G. Miranda, J. Hoebeke, J. M. Marquez-Barja, and I. Moerman, "Hardware Efficient Clock Synchronization across Wi-Fi and Ethernet Based Network Using PTP," *IEEE Transactions on Industrial Informatics*, pp. 1–1, 2021. [Online]. Available: doi.org/10.1109/TII.2021.3120005

[14] "Ieee standard for local and metropolitan area networks - virtual bridged local area networks amendment 12: Forwarding and queuing enhancements for time-sensitive streams," *IEEE Std 802.1Qav-2009*, pp. C1–72, 2010. [Online]. Available: doi.org/10.1109/IEEESTD.2009.5375704

[15] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic," *IEEE Std 802.1Qbv-2015*, pp. 1–57, 2016. [Online]. Available: 10.1109/IEEESTD.2016.8613095

[16] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption," *IEEE Std 802.1Qbu-2016*, pp. 1–52, 2016. [Online]. Available: doi.org/10.1109/IEEESTD.2016.7553415

[17] "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing," *IEEE Std 802.1Qci-2017*, pp. 1–65, 2017. [Online]. Available: doi.org/10.1109/IEEESTD.2017.8064221

[18] "IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability," *IEEE Std 802.1CB-2017*, pp. 1–102, 2017. [Online]. Available: doi.org/10.1109/IEEESTD.2017.8091139

[19] "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*, pp. 1–208, 2018. [Online]. Available: doi.org/10.1109/IEEESTD.2018.8514112

[20] J. Haxhibeqiri, X. Jiao, E. Municio, J. M. Marquez-Barja, I. Moerman, and J. Hoebeke, "Bringing Time-Sensitive Networking to Wireless Professional Private Networks," *Wireless Personal Communications*, pp. 1–17, 2021. [Online]. Available: doi.org/10.1007/s11277-021-09056-0

[21] S. B. H. Said, Q. H. Truong, and M. Boc, "SDN-Based Configuration Solution for IEEE 802.1 Time Sensitive Networking (TSN)," *SIGBED Rev.*, vol. 16, no. 1, p. 27–32, Feb. 2019. [Online]. Available: doi.org/10.1145/3314206.3314210

[22] T. Kobzan, I. Blöcher, M. Hendel, S. Althoff, A. Gerhard, S. Schriegel, and J. Jasperneite, "Configuration Solution for TSN-based Industrial Networks utilizing SDN and OPC UA," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2020, pp. 1629–1636. [Online]. Available: doi.org/10.1109/ETFA46521.2020.9211897

[23] S. S. Andrej Friesen and A. Biendarra, "PROFINET over TSN Guideline Version 1.1." [Online]. Available: www.profibus.com/technology/profinet

[24] K. A. Nsiah, K. Alkhouri, and A. Sikora, "Configuration of Wireless TSN Networks," in *2020 IEEE 5th International Symposium on Smart and Wireless Systems within the Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS)*, 2020, pp. 1–5. [Online]. Available: doi.org/10.1109/IDAACS-SWS50031.2020.9297066

[25] M. Böhm, J. Ohms, M. Kumar, O. Gebauer, and D. Wermser, "Time-sensitive software-defined networking: a unified control-plane for TSN and SDN," in *Mobile Communication-Technologies and Applications; 24. ITG-Symposium*. VDE, 2019, pp. 1–6. [Online]. Available: ieeexplore.ieee.org/abstract/document/8731777

[26] S. Bhattacharjee, K. Katsalis, O. Arouk, R. Schmidt, T. Wang, X. An, T. Bauschert, and N. Nikaein, "Network Slicing for TSN-Based Transport Networks," *IEEE Access*, vol. 9, pp. 62 788–62 809, 2021. [Online]. Available: doi.org/10.1109/ACCESS.2021.3074802

[27] M. Gutiérrez, A. Ademaj, W. Steiner, R. Dobrin, and S. Punnekkat, "Self-configuration of IEEE 802.1 TSN networks," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2017, pp. 1–8. [Online]. Available: doi.org/10.1109/ETFA.2017.8247597

[28] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling Fog Computing for Industrial Automation Through Time-Sensitive Networking (TSN)," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 55–61, 2018. [Online]. Available: doi.org/10.1109/MCOMSTD.2018.1700057

[29] M. Pahlevan, J. Schmeck, and R. Obermaisser, "Evaluation of TSN Dynamic Configuration Model for Safety-Critical Applications," in *2019 IEEE Intl Conf on Parallel Distributed Processing with Applications, Big Data Cloud Computing, Sustainable Computing Communications, Social Computing Networking (ISPA/BDCloud/SocialCom/SustainCom)*, 2019, pp. 566–571. [Online]. Available: doi.org/10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00086

[30] M.-T. Thi, S. B. H. Said, and M. Boc, "SDN-Based Management Solution for Time Synchronization in TSN Networks," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1. IEEE, 2020, pp. 361–368. [Online]. Available: doi.org/10.1109/ETFA46521.2020.9211923

[31] D. Bruckner, M.-P. Stănică, R. Blair, S. Schriegel, S. Kehrer, M. Seewald, and T. Sauter, "An introduction to OPC UA TSN for industrial communication systems," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1121–1131, 2019. [Online]. Available: doi.org/10.1109/JPROC.2018.2888703

[32] U. Alliance, "Digital Enhanced Cordless Telecommunications (DECT) Ultra Low Energy (ULE)." [Online]. Available: www.etsi.org/deliver/etsi_ts/102900_102999/10293901/01.03.01_60/ts_10293901v010301p.pdf

[33] N. G. Nayak, F. Dürr, and K. Rothermel, "Time-sensitive software-defined network (TSSDN) for real-time applications," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, 2016, pp. 193–202. [Online]. Available: doi.org/10.1145/2997465.2997487

[34] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury, "Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2018. [Online]. Available: doi.org/10.1109/COMST.2018.2869350

[35] D. Hellmanns, L. Haug, M. Hildebrand, F. Dürr, S. Kehrer, and R. Hummen, "How to Optimize Joint Routing and Scheduling Models for TSN Using Integer Linear Programming," 2021. [Online]. Available: doi.org/10.1145/3453417.3453421

[36] J. Haxhibeqiri, A. Seferagic, R. V. Bhat, I. Moerman, and J. Hoebeke, *Tighter Application-Network Interfacing to Drive Innovation in Networked Systems*. New York, NY, USA: Association for Computing Machinery, 2021, p. 53–57. [Online]. Available: doi.org/10.1145/3472727.3472801

[37] S. Schriegel, T. Kobzan, and J. Jasperneite, "Investigation on a distributed SDN control plane architecture for heterogeneous time sensitive networks," *IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, vol. 2018-June, pp. 1–10, 2018. [Online]. Available: doi.org/10.1109/WFCS.2018.8402356

[38] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network configuration protocol (NETCONF)," 2011. [Online]. Available: www.hjp.at/doc/rfc/rfc6241.html

[39] J. Schönwälder, M. Björklund, and P. Shafer, "Network configuration management using NETCONF and YANG," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 166–173, 2010. [Online]. Available: doi.org/10.1109/MCOM.2010.5560601

[40] J. Haxhibeqiri, P. H. Isolani, J. M. Marquez-Barja, I. Moerman, and J. Hoebeke, "In-Band Network Monitoring Technique to Support SDN-Based Wireless Networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 627–641, 2021. [Online]. Available: doi.org/10.1109/TNSM.2020.3044415

[41] J. Haxhibeqiri, I. Moerman, and J. Hoebeke, "Low overhead, fine-grained end-to-end monitoring of wireless networks using in-band telemetry," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019, pp. 1–5. [Online]. Available: doi.org/10.23919/CNSM46954.2019.9012678

[42] Z. Kang, R. Canady, A. Dubey, A. Gokhale, S. Shekhar, and M. Sedlacek, "A Study of Publish/Subscribe Middleware Under Different IoT Traffic Conditions," in *Proceedings of the International Workshop on Middleware and Applications for the Internet of Things*, ser. M4IoT'20. New York, NY, USA: Association for Computing Machinery, 2020, p. 7–12. [Online]. Available: doi.org/10.1145/3429881.3430109

[43] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks," in *Proceedings of the 24th International Conference on Real-Time Networks and Systems - RTNS '16*, vol. 19-21-Octo. New York, New York, USA: ACM Press, 2016, pp. 183–192. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2997465.2997470

[44] A. C. T. d. Santos, B. Schneider, and V. Nigam, "TSNSCHED: Automated Schedule Generation for Time Sensitive Networking," in *2019 Formal Methods in Computer Aided Design (FMCAD)*, 2019, pp. 69–77. [Online]. Available: doi.org/10.23919/FMCAD.2019.8894249

[45] N. G. Nayak, F. Durr, and K. Rothermel, "Incremental Flow Scheduling and Routing in Time-Sensitive Software-Defined Networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2066–2075, may 2018. [Online]. Available: doi.org/10.1109/TII.2017.2782235

[46] J. Falk, F. Durr, and K. Rothermel, "Exploring Practical Limitations of Joint Routing and Scheduling for TSN with ILP," in *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, aug 2018, pp. 136–146. [Online]. Available: doi.org/10.1109/RTCSA.2018.00025

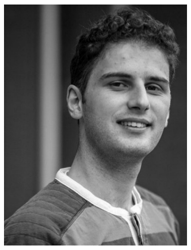
[47] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000. [Online]. Available: doi.org/10.1145/354871.354874

[48] W. Vandenberghe, B. Vermeulen, P. Demeester, A. Willner, S. Papavassiliou, A. Gavras, M. Sioutis, A. Quereilhac, Y. Al-Hazmi, F. Lobillo *et al.*, "Architecture for the heterogeneous federation of future internet experimentation facilities," in *2013 Future Network & Mobile Summit*. IEEE, 2013, pp. 1–11. [Online]. Available: ieeexplore.ieee.org/abstract/document/6633558

[49] Avila-Campos, Pablo and Haxhibeqiri, Jetmir and Moerman, Ingrid and Hoebeke, Jeroen, "Impactless Beacon-Based Wireless TSN Association Procedure," *18th IEEE International Workshop on Factory Communication Systems - Proceedings, WFCS*, 2022. [Online]. Available: doi.org/10.1109/WFCS53837.2022.9779186

Gilson Miranda Jr. holds B.Sc. and M.Sc. degrees in Computer Science from Federal University of Lavras (UFLA), Brazil. In 2017 started his PhD in Computer Science at the Federal University of Minas Gerais (UFMG), Brazil. Now is pursuing a Joint PhD in Applied Engineering at the University of Antwerp, Belgium, where he is carrying his research with IDLab. His main research interests are programmable Time-Sensitive Networks, wireless networks, and machine learning applied to network management.

Esteban Municio Esteban Municio received his Ph.D. degree from the University of Antwerp and imec (Belgium) in 2020. He then continued in imec as postdoctoral researcher for two years. Since January 2022, he has been with i2CAT, where currently he is senior researcher at the AI-driven Systems group. His research interests are in the field of programmable open networks, Time-Sensitive Networking and ultra-reliable Industrial IoT.



Jetmir Haxhibeqiri received the Masters degree in Engineering (information technology and computer engineering) from RWTH Aachen University, Germany (2013). In 2019, he obtained a Ph.D. in Engineering Computer Science from Ghent University with his research on flexible and scalable wireless communication solutions for industrial warehouses and logistics applications. Currently, he is a senior researcher in the Internet Technology and Data Science Lab (IDLab) of Ghent University and imec. His current research interests include wireless communications technologies (IEEE 802.11, IEEE 802.15.4e, LoRa) and their application, IoT, wireless time-sensitive networking, in-band network monitoring and wireless network management.

Jeroen Hoebeke is an associate professor in the Internet Technology and Data Science Lab of Ghent University and imec. He is conducting and coordinating research on deterministic and time-sensitive wireless communication, wireless network management, tighter application-network integration, (industrial) IoT connectivity, and embedded communication stacks. This expertise has been applied in various application domains such as logistics, Industry 4.0, building automation, healthcare, and animal monitoring. He is particularly active in nationally funded projects as well as in defining, executing, and managing such projects. He has also been involved in several EU research-funded projects and is the author or co-author of more than 200 publications in international journals or conference proceedings.

Ingrid Moerman received her degree in Electrical Engineering (1987) and the Ph.D. degree (1992) from the Ghent University, where she became a part-time professor in 2000. She is a staff member at IDLab, a core research group of imec with research activities embedded in the universities of Ghent and Antwerp. She coordinates the research activities on intelligent Wireless Networking (iWiNe) at Ghent University, leading a team of more than 30 researchers. She is Program Manager of the 'Deterministic Networking' track, part of the CONNECTIVITY program at imec, where she coordinates research activities on end-to-end wired/wireless networking solutions for professional and mission-critical applications. She has coordinated several FP7/H2020 projects (CREW, WiSHFUL, eWINE, ORCA) and she is involved in many national and H2020 and Horizon Europe projects related to connected vehicles and 5G/6G (Smart Highway, CONCORDA, 5G-MOBIX, 5G-CARMEN, 5G-Blueprint, DEDICAT-6G, HEXA-X II, TrialsNet and 6G-SHINE). She participated in the prestigious DARPA Spectrum Collaboration Challenge (SC2) as the lead of Team SCATTER. This team has been awarded two prizes of 750,000 USD each in Phase 1 (2017) and Phase 2 (2018) of the DARPA SC2 competition and was one of the 10 finalists at the DARPA SC2 championship event organized at Mobile World Congress in Los Angeles (US) in October 2019 (https://www.darpa.mil/news-events/2019-09-10).

Johann Marquez-Barja is a Professor at the University of Antwerp (Rank 7th in the Times Higher Education Under 50) and also a Professor in IMEC Research Centre (Worldwide leading in Nanotechnologies and Digital solutions), Belgium. Currently, he is leading the Flexible & Programmable Networks Group at IDLab/imec Antwerp. Previously he led the Wireless Cluster. He was/is involved in more than 20 European research projects ranging from long-range to high-capacity radio and networking technologies. He is currently the technical coordinator of the 5G Blueprint project, which focuses on improved 5G cross-border networks to enable the teleoperation of vehicles and vessels. He is a member of IEEE Standards Association, Association for Computing Machinery (ACM), a Fellow of European Association for Innovation (EAI), a Senior Member of the IEEE Communications Society, IEEE Vehicular Technology Society, and IEEE Education Society, where he participates on the board of the Standards Committee. His main research interests are 5G advanced architectures, including edge computing; flexible and programmable 5G and 6G end-to-end networks; IoT communications, and applications. He is also interested in smart mobility and smart city deployments. He leads the Citylab Smart City testbed and the Smart Highway testbed, located in Antwerp, Belgium. He is also active in education development and actively involved in different research actions to enhance engineering education, particularly remote experimentation for online labs. He has given several keynotes and invited talks at various major events, received 30 awards in his career so far, and co-authored more than 200 articles. He is also serving as Editor and Guest editor for different International Journals, as well as participating in several Technical Programme and Organizing Committees for several worldwide conferences/congresses.