UNICO 5G I+D

TIMING

Towards a smart and efficient telecom infrastructure meeting current and future industry needs (TIMING SP-2)
(Ref. TSI-063000-2021-148)

Deliverable D1.1

Year 1 report on component development, integration, testing and validation

Editor S. Spadaro (UPC)

Contributors ELI, IKL, SED, UPC, ABB, Telefonica

Version 1.5

Date 31 Jan-2024

Distribution PUBLIC (PU)

DISCLAIMER

REVISION HISTORY

Revision	*Date*	*Responsible*	*Comment*
1.0	*July, 14, 2023*	*UPC*	*Tentative Table of Contents*
1.1	*December, 12, 2023*	*UPC*	*Responsible Sections assignment*
1.2	*January, 19, 2024*	*UPC*	*Integrated version*
1.3	*January, 25, 2024*	*UPC*	*2nd Integrated version*
1.4	*January, 30, 2024*	*UPC*	*Revised version*
1.5	*January, 31, 2024*	*UPC*	*Final Version*

LIST OF AUTHORS

Partner ACRONYM	***Partner FULL NAME***	***Name & Surname***
UPC	*Universitat Politècnica de Catalunya*	*S. Spadaro, L. Velasco, F. Agraz, M. Ruiz, J. Comellas, D. Careglio, M. Cabrera, J. Villares, J. Vidal, O. Muñoz*
IKL	*Ikerlan*	*G. Ros, R. Torrego*
ELI	*E-Lighthouse*	*J. Moreno, P. Pavón Mariño, E. Férnandez Sánchez*
SED	*Safran Electronics & Defense*	*C. Arias, J. Sánchez*
TID	*Telefónica I+D*	*L. M. Contreras, M. Blanco, J. Folgueira*
ABB	*Asea Brown Bovery*	*L. Gonzalez*

GLOSSARY

Abbreviations/Acronym	*Description*
AGV	*Automated Guided Vehicle*
AP	*Access Point*
API	*Application Programming Interface*
CAN	*Controller Area Network*
CM	*Connectivity Manager*
CNC	*Central Network Controller*
DT	*Digital Twin*
KPI	*Key Performance Indicator*
JSON	*JavaScript Object Notation*
LIDAR	*Light Detection and Ranging*
MCS	*Modulation Coding Scheme*
NBI	*NorthBound Interface*
OFDM	*Orthogonal Frequency-Division Multiplexing*
PTP	*Precision Time Protocol*
SBI	*SouthBound Interface*
STA	*Station in a Wi-Fi network*
TSN	*Time Sensitive Networks*
VHDL	*Wi-Fi TSN Node Hardware*
WFS	*Wi-Fi Scheduler*

EXECUTIVE SUMMARY

The aim of this deliverable is to report about the status of the different components/modules of the overall TIMING architecture in support of E2E services, defined in the framework of SP1 project. In particular, it reports the status of the prototyping of the Wi-Fi and Ethernet TSN nodes, the TSN controller, the TSN Connectivity Manager, the scheduling algorithms for the Wi-Fi node, TSN models to be used by the Digital Twin (DT) to estimate the KPIs, metro network solution, and the industrial application.

Moreover, this deliverable contains a preliminary version of the operational workflow for the service provisioning over the E2E infrastructure, putting emphasis on the interactions among the different components to materialize the required resources configuration to support the E2E services with the required KPIs.

Finally, this deliverable also contains the preliminary integration tests conducted so far, with special emphasis on the data plane interconnection tests between TSN Wi-Fi node and the TSN Ethernet node.

TABLE OF CONTENTS

1 INTRODUCTION

This deliverable reports, firstly, the ongoing status of implementation of the different components/modules that have been defined in the architecture design, reported in the SP-1 deliverables. More details about the overall architecture can be found in deliverable D1.2 [1] and D1.3 [2]. The implementation of the different modules as well as their evaluation represents the required step prior to their integration to materialize the TIMING architecture properly designed to support E2E service provisioning. The different components that compose the architecture can be listed as:

- Wi-Fi-based TSN nodes
- Scheduling algorithms for Wi-Fi TSN
- Ethernet TSN nodes
- TSN SDN controller
- Connectivity Manager
- TSN models/DT
- Optical transport/metro SDN Controller
- Industrial application.

For each component, this deliverable reports the current status of the development, as well some stand-alone tests to validate the implementation. Additionally, this deliverable also reports some preliminary integration tests, mostly at the data plane level, that is, related to data plane connectivity between the Wi-Fi node and the Ethernet TSN node.

The overall integration tests will be reported in the upcoming deliverable D1.2 of SP2 project.

2 TIMING ARCHITECTURE COMPONENTS

2.1 TSN WI-FI NODE

2.1.1 Description

The SW/HW architecture of the WI-FI TSN Node can be divided in three sections as shown Figure 2-1.

- **Wi-Fi TSN Node's hardware (VHDL)**, including key components such as physical communications interfaces (Ethernet TSN and Wi-Fi TSN), a PTP Hardware clock, and a RT traffic translation entity. The Ethernet TSN features at least one physical port, supporting IEEE 802.1 standards such as IEEE 802.11AS for PTP and IEEE 802.1Qbv for scheduled and mixed-critical traffic. The Wi-Fi TSN modem employs a w-SHARP implementation, incorporating time-sensitive communication and synchronization techniques as in the Wired TSN interface.
- **The main processor** who operates a general-purpose operating system, hosting the primary control application, hardware and control access drivers, a TCP/IP network stack, and various applications for Wi-Fi TSN node operation. These applications consist of a PTP stack for IEEE 802.1AS compliance, a controller interface for external TSN controller communication, and a scheduling algorithm interface for wireless scheduling methods.
- **The real-time co-processor** who hosts the Wi-Fi TSN modem driver, RT software program, and components governing precise-time operations of the Wi-Fi TSN modem. The Wireless TSN driver facilitates frame communication with the Wi-Fi TSN modem, offering a control interface for the scheduling algorithm and TSN controller.

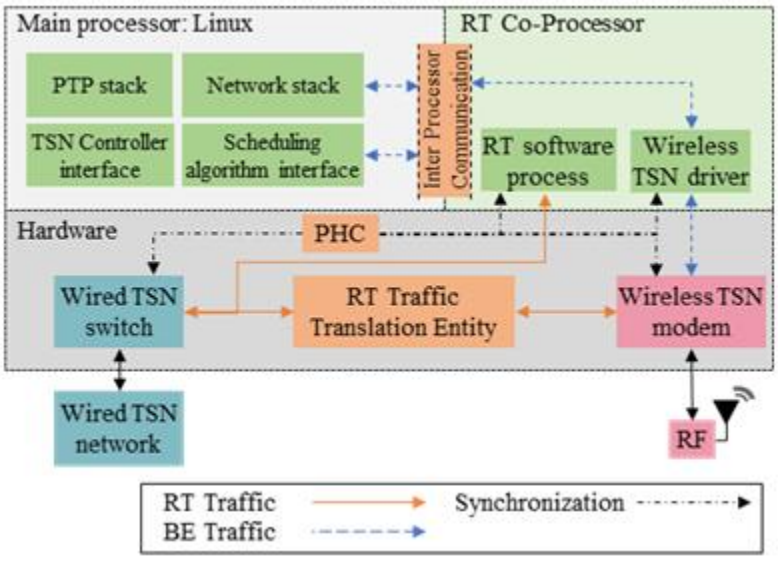


Figure 2-1: Block diagram of the Wi-Fi TSN node HW/SW architecture.

With respect to the status reported in SP1 D1.3 [2], work has been done specially in two main topics:

- Provide fast **reconfiguration of the Wi-Fi-TSN nodes**: Every time the SDN-TSN controller or the Wireless Flow Scheduler demands a change in the configuration, the capability to

receive the new specific TSN configuration and be able to forward it to the connected STAs and apply it as fast as possible.
- Provide **statistics/telemetry/capabilities** to the SDN-TSN controller and the Wireless Flow Scheduler.

Along with these, the first integration tests between Ikerlan's WiFi-TSN nodes and Safran's TSN switches are also reported in section 5.

2.1.2 Report on the Current Status

2.1.2.1 Current status of the implementation

SP1 derivable D1.3 [2], section 2.2.1 defined three stages for the development of the new control software:

- A first stage where the different features (publications of statistics, reception of configuration and propagation of configuration) will be implemented through configuration files. The remote access either from the SDN-TSN Controller or the Wireless Flow Scheduler to change the configuration files or read statistics will be done via FTP or SSH connection. The retransmission of the new configuration from the AP to the STAs is also done via SSH and FTP.
- A second stage, where the configuration files from the SDN-TSN Controller and the Wireless Flow Scheduler are received, and the gathered statistics are offered via an API based in an IP/Socket protocol. Besides, the retransmission of the new configuration from the AP to the STAs will be included in the beacon, at the beginning of each super-frame, thus speeding un the reconfiguration process.
- A third stage, where the information to/from the SDN-TSN Controller will continue being received/sent via an API based in a IP/Socket protocol, but where the Wireless Flow Scheduler will be integrated in the real-time software of the TSN Wi-Fi node.

At this point, the first stage is practically completed, and the second stage is ongoing.

On the one hand, from the fast reconfiguration of the WiFi-TSN nodes point of view, the development of two APIs, with the aim of receiving the configuration information, is ongoing. One API would be aimed for the SDN-TSN controller, and the other one for the Wireless Flow Scheduler. The API being developed for the SDN-TSN controller is a Flask based solution as its reconfiguration speed requisites are not very demanding. For the Wireless Flow Scheduler an IP/socket based solutions is being developed and tested in order to achieve the highest possible speed.

In addition, some software modifications addressing the possibility of transmitting this configuration information from the AP to the STAs in the beacon are being implemented. In this way, the configuration information would be transmitted at the beginning of each super-frame, offering a higher reconfiguration speed compared to FTP/SSH based transmissions. The new information being sent in the beacon frame can be seen in Figure 2-2.

```
struct SHARP_beacon_frame_data_str
{
    uint8_t DST_ID;
    uint8_t ORIG_ID;
    uint8_t reserved;
    uint8_t valid_flag;
    uint32_t ID;
    uint32_t timestamp;
    uint32_t timestamp_s;
    struct superframe_info sframe_info;
}__attribute__((packed));

struct superframe_info {
    uint16_t TSN_cycle;
    uint8_t STD_period;
    uint8_t DL_start;
    uint8_t N_DL_flows;
    uint8_t N_UL_flows;
    struct Sharp_Flow DL_Flows[MAX_N_DL_flows];
    struct Sharp_Flow UL_Flows[MAX_N_UL_flows];
} __attribute__((packed));;

 struct Sharp_Flow{
    char TSN_DST[12];
    char TSN_ORIG[12];
    uint8_t TSN_PCP;
    uint8_t TSN_VLAN_ID;
    char w_SHARP_header[8];
    uint8_t length;
    uint8_t MCS;
    uint8_t frame_type;
    uint8_t needs_ACK;
    uint8_t STA;
} __attribute__((packed));;
```

Figure 2-2: New beacon data structure, adding the super-frame configuration.

Regarding the provision of statistics/telemetry/capabilities to other devices in the TIMING architecture, some changes are being done in the control software of both processors of the Wi-Fi TSN Node. Most of the statistics are collected in the RT processor so it has been necessary to enable the transmission of this information from this processor to the general processor by using the inter-processor communication module (remoteproc) as shown in Figure 2-3. Once transmitted to the general-purpose processor, they are compiled in a file so that they can be consumed from the API for the Wireless Scheduler and the SDN-TSN Controller.

```
static void SHARP_AP_UL_Data_Rx(void *pvParameters)
{
    BaseType_t xQueue_ret;
    uint32_t xQUEUE_buffer_input;
    struct SHARP_irq_data* SHARP_irq_data_ptr;
    uint32_t* Desc_src_ptr;
    uint8_t DL_STA_Rx_end_ptr_alloc[8U+DESC_RX_END_BYTES*4U];
    struct DL_STA_Rx_end_str* DL_STA_Rx_end_ptr;
    static int aux=0;
    DL_STA_Rx_end_ptr = (struct DL_STA_Rx_end_str*) DL_STA_Rx_end_ptr_alloc;
    uint32_t log[4];

    for(;;)
    {
        xQueue_ret = xQueueReceive(xQueue_AP_UL_Data_Rx, (uint32_t*)&xQUEUE_buffer_input, portMAX_DELAY);

        if (xQueue_ret == pdTRUE){

            SHARP_irq_data_ptr = (struct SHARP_irq_data*) &xQUEUE_buffer_input;
            Desc_src_ptr = (uint32_t*)(SHARP_DESC_WR_MEM_BASEADDR+SHARP_irq_data_ptr->Desc_end_ADDR);
            uint32_t No_bytes = 312;
            SHARP_memcpy_32((uint32_t*)DL_STA_Rx_end_ptr,Desc_src_ptr,No_bytes);

            if (DL_STA_Rx_end_ptr->Desc_Rx_end.status == 0){
                aux++;

                if (aux==20000){

                    enum rproc_msg_types_e rproc_msg_types = SET_LOG;
                    enum rproc_msg_types_e rproc_msg_set_log_types_e = LOG_RSSI;
                    log[0] = rproc_msg_types;
                    log[1] = rproc_msg_set_log_types_e;
                    log[2] = DL_STA_Rx_end_ptr->Desc_Rx_end.RSSI;
                    log[3] = DL_STA_Rx_end_ptr->Desc_Rx_end.MCS;

                    taskENTER_CRITICAL();
                    lept.dest_addr = 0x00000400;
                    rpmsg_send(&lept, log, 16);
                    taskEXIT_CRITICAL();
                    aux=0;
                }

            }

        }
    }
}
```

Figure 2-3: First version of the task of transmitting statistics from the RT processor to the general task processor.

2.1.2.2 Stand-alone preliminary tests performed

In order to evaluate the current reconfiguration speed, the time between the application of the new configuration to an AP and the reconfiguration of both AP and STA has been measured.

The measured time is around 20 seconds.

Actions	**Time**
Detect new TSN configuration file	1 ms
Transmit/Receive TSN configuration File	500 ms
Apply new TSN configuration File	15s
Syncronize to transmit again	5s
TOTAL	**≈ 20s**

Currently the run-time reconfiguration capability of the Wi-Fi TSN Node is limited due to the model based in configuration files and transmission of the configuration via FTP/SSH protocols between AP and STA. Most of the time is taken applying the new configuration file, due to the need to parse the file from the general-purpose processor to the RT processor, and from there, to the HW.

2.2 SCHEDULING ALGORITHMS FOR WI-FI TSN

2.2.1 Description

The Wi-Fi scheduler (WFS) interacts with the Wi-Fi TSN node at regular superframe intervals. On one hand, the Wi-Fi TSN node reports details about the active flows, including comprehensive information such as traffic characteristics, priority type, direction (uplink or downlink), queue size for asynchronous traffic and bit rate for isochronous traffic. Additionally, the Wi-Fi TSN node provides environmental information, particularly regarding the state of the channel. Provided with this information, the scheduler is tasked with making decisions about the structure of Wi-Fi frames. The global scheduler algorithm, integrated into the TSN Wi-Fi node, encompasses several modules, as depicted in Figure 2-4 and summarized below:

- TSN Wi-Fi Windows Designer. At the TSN Wi-Fi node, each isochronous TSN flow is allocated into a periodic time window by the connectivity manager, considering the required throughput and the worst-case channel state. The WFS, depending on the channel state, optimizes the pre-assigned time window in the Wi-Fi link, reducing the size of the time window assigned to the isochronous flows if the channel state allows it and thus freeing slots that can be reused by other non-isochronous and lower priority flows. In this sense, the advanced window designs described in D1.4 (SP1) will not be implemented in SP2 for the sake of compatibility with the Wi-Fi node hardware and the regular TSN architecture adopted in Timing, instead, the TSN traffic corresponding to a single flow is transmitted in the pre-allocated time window at the maximum speed allowed by the channel, which is equivalent to reducing the pre-allocated window to the here named Wi-Fi window.
- DL/UL Splitter for UL and DL distribution of the slots not devoted to isochronous traffic. It computes dynamically the number of slots assigned to DL and UL by applying a reinforcement learning based algorithm which takes decisions depending on the aggregated DL and UL queues sizes.
- DL Asynchronous Traffic Scheduler: Determines the flows served in the DL prioritizing first the asynchronous traffic with quality of service over the BE traffic without QoS.
- UL Asynchronous Traffic Scheduler: Analogously to the DL scheduler, it decides the flows to be served in the uplink considering the established priorities.
- Frame Builder: It builds the frame based on the designed TSN Wi-Fi windows for isochronous traffic and on the DL&UL scheduler decisions.

The flow chart that completely encompasses the scheduler is shown in Figure 2-4.

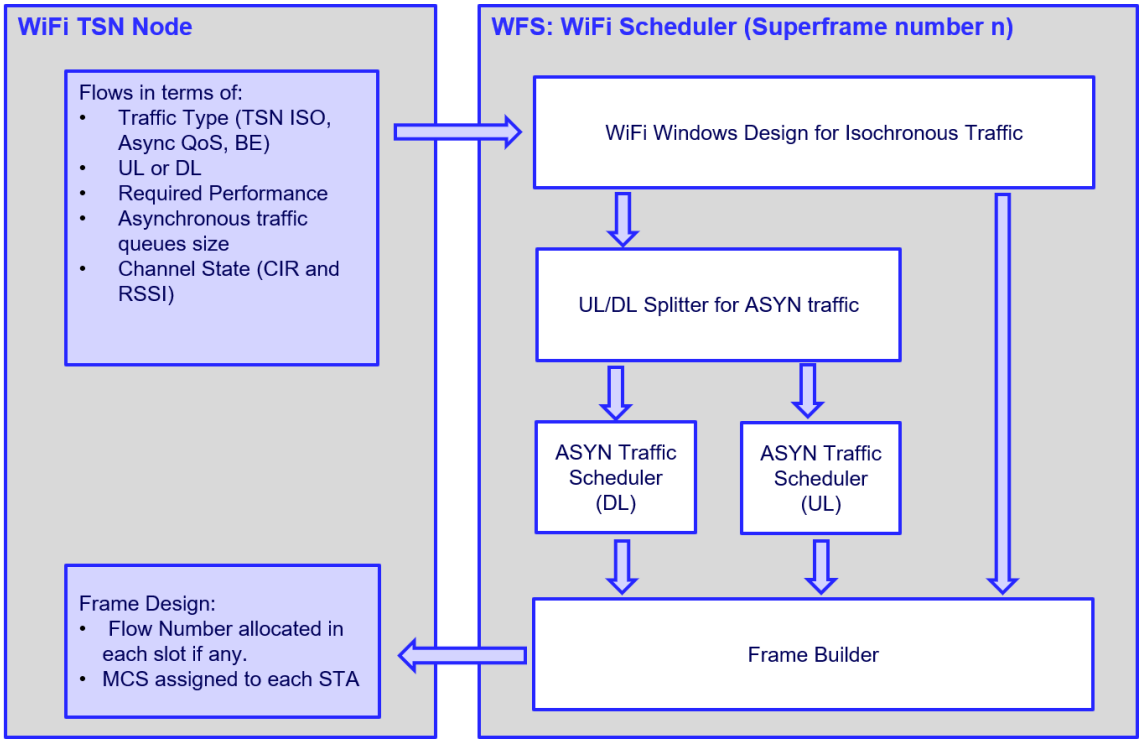


Figure 2-4: WiFi Scheduler (WFS) flowchart

Figure 2-5 shows an example in which in the downlink part of the frame, flows have been allocated into the different slots. The Wi-Fi time windows assigned to isochronous traffic are symbolized in yellow.

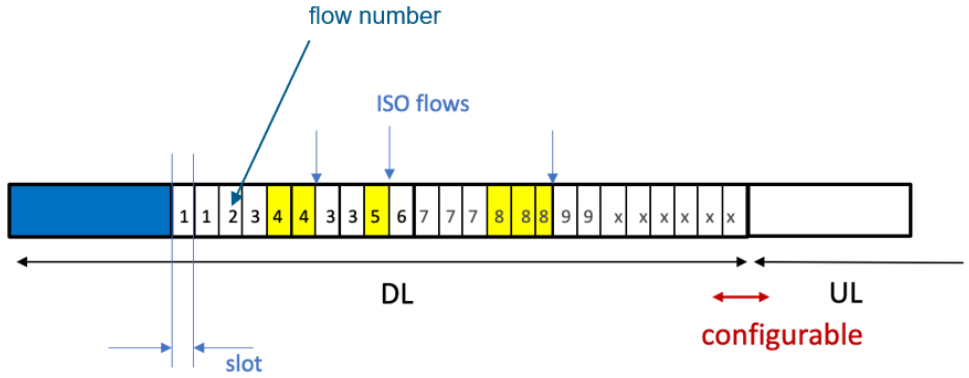


Figure 2-5: Frame Design Example

2.2.2 APIs design to integrate the scheduler in the Wi-Fi Node

In the API design to integrate the WFS into the TSN Wi-Fi Node some steps have to be defined or planned previously, as listed below.

1. API Design: To integrate the WFS into the TSN node, an API will be designed to manage the interface between both. Progress in programming the API will be carried out in three successive stages.
    a. In the first stage a socket in Matlab will be programmed. In the first integration test a basic configuration and communications block will work in Matlab

dynamically providing a frame design every 5 sec. approximately, which is determined by the limitations of the Matlab socket.

b. In the second stage, the same development will be carried out but programming in C, expecting to obtain a higher periodicity in the frame structure updating process.

c. In the most advanced level (third stage), the C-programmed API will be compiled into the Linux program, embedding the API locally within the SHARP Wi-Fi TSN card. This approach is intended to achieve the fastest speed in the periodicity of frame design.

The parameters that will be exchanged through the implemented API are those given in tables I and II in deliverable SP-1 D1.3, specifically in Section 2.3.1.

2. Wi-Fi time window design for isochronous traffic: Regarding the isochronous flows, the connectivity manager will orchestrate the route as well as the time window assigned for its transmission in each of the network elements. Based on this design, the TSN Wi-Fi node will inform the WFS of the periodicity and the start and end time stamps of the assigned window, initially designed to support the lowest MCS (channel state worst-case). In the initial phase of the scheduling algorithm, if the channel state is better than the worst-case scenario, the MCS will be increased and the time windows for the Wi-Fi segment will be adjusted. This involves allocating the corresponding flow into a smaller time interval than the initial one, and as a result it will be possible to opportunistically accommodate lower-priority and asynchronous traffic in the free time intervals. The WFS will define the assigned time interval in slots (OFDM symbols), serving as the temporal granularity unit within the Wi-Fi segment.
   In any case, it must be guaranteed that the transmission of the assigned flows, in each bridge, does not exceed the time windows assigned by the connectivity manager. If necessary, while Wi-Fi windows are prepared, the information can be buffered in the STA devices (UL) or the AP (DL) without violating the previous condition.

3. Real-Time DL/UL split: The Real-Time DL/UL splitter will compute the size of the aggregated DL queue, i.e. the summation of the sizes of all the asynchronous DL queues (traffic with service quality and BE if there are), considering different levels of priority and the size of the aggregated UL queue. Then it will also compute the number of free slots in the frame, once isochronous flows have been allocated in the frame, and by applying a RL based decision it will deliver the number of slots allocated for asynchronous traffic in RT-DL and the number of slots allocated for asynchronous traffic in RT-UL. Alternatively, it will decide the frontier of the RT-DL and RT-UL subframes.

4. Scheduler for Real-Time DL frame [DOWNLINK]: The Real-Time DL scheduler allocates slots among the Downlink (DL) flows associated with quality of service using a largest-weighted-delayed-first scheduler (LWDF) algorithm. This algorithm leverages channel state information and buffers occupation to determine the number and specific slots assigned to each flow. If there are remaining free slots, they are then distributed among Best Effort (BE) flows using a straightforward Round-Robin algorithm.

5. Scheduler for Real-Time UL frame [UPLINK]: In UL, the same scheduler is used, as in DL, but in this case a guard time and a preamble must be inserted before the slots corresponding to the different users or STAs.

6. Channel modeling: The channel model for scheduler design and simulation has to be validated providing models for pathloss in terms of distances, pathloss exponent and average received power and for fading profile in terms of PDP, Delay spread, Rayleigh or Rice Line of Sight, Doppler spread and coherence time.

7. SHARP Frame definition (RT subframe): RT frame definition for scheduler design and simulation has to be validated, providing values for bandwidth, OFDM symbol length, slot time, number of carriers for data, MCS modes, inter-symbol spacing, and PHY&MAC overhead in UL and in DL.

Table 1 shows the input/output variables associated to the different blocks that make up the scheduler and

Model	**Parameter**	**Value**
Channel Pathloss	Distance	TBD
	Exponent	TBD
	Average received power	TBD
Multipath Fading	Delay Spread (rms)	89 ns (Ch1) - 29 ns (Ch2)
	NLOS	Yes/No
	Doppler spectrum	TBD
	Coherence Time	30 msec.
Frame Structure	Bandwidth	20 MHz
	OFDM symbol length	4 s (0.8 s + 3.2 s)
	Slot Time	OFDM symbol
	Number of data subcarriers	48
	MCS	0…7
	PHY overhead (DL)	20 s
	PHY overhead (UL)	4 s
	MAC overhead (DL)	TBD
	MAC overhead (UL)	TBD
	IFS (Interframe Spacing)	TBD

Table 2 shows the different parameters needed for both integration and simulation purposes.

Block	**Inputs**	**Outputs**

1 Wi-Fi time window design for isochronous traffic	Global Information: ● Number of isochronous flows. Information per flow: ● Channel State (Channel Impulse Response CIR and Received Signal Strength Indicator RSSI) ● Pre-assigned Time window (Initial and final time stamps) ● Direction (UL or DL) ● STA identifier or ID ● Periodicity of packets ● Maximum PER	Information per flow: ● MCS ● Assigned slots
2 Real-Time DL/UL split	Global Information: ● Number of free slots in the frame. ● Number of asynchronous flows. Information per flow: ● Direction (UL or DL) ● Queue size	Global Information: ● Number of slots assigned to asynchronous flows in the DL. ● Number of slots assigned to asynchronous flows in the UL.
3 Scheduler for Real-Time DL frame	Global Information: ● Number of slots assigned to asynchronous flows in the DL. ● Number of asynchronous DL flows. Information per DL flow: ● Channel State (CIR and RSSI) ● STA ● Maximum PER ● Maximum latency	Information per flow: ● MCS ● Assigned slots
4 Scheduler for Real-Time UL frame	Global Information:	Information per flow: ● MCS

	• Number of slots assigned to asynchronous flows in the UL. • Number of asynchronous UL flows. Information per UL flow: • Channel State (CIR and RSSI) • STA • Maximum PER • Maximum latency	• Assigned slots
5 Frame Builder	<u>Global Information:</u> • Number of flows Information per flow: • MCS • Assigned slots	<u>Global Information:</u> • Frame structure

Table 1: Input and output variables at the WFS functional blocks

Model	**Parameter**	**Value**
Channel Pathloss	Distance	TBD
	Exponent	TBD
	Average received power	TBD
Multipath Fading	Delay Spread (rms)	89 ns (Ch1) - 29 ns (Ch2)
	NLOS	Yes/No
	Doppler spectrum	TBD
	Coherence Time	30 msec.
Frame Structure	Bandwidth	20 MHz
	OFDM symbol length	4 s (0.8 s + 3.2 s)
	Slot Time	OFDM symbol
	Number of data subcarriers	48
	MCS	0…7
	PHY overhead (DL)	20 s
	PHY overhead (UL)	4 s

MAC overhead (DL)	TBD
MAC overhead (UL)	TBD
IFS (Interframe Spacing)	TBD

Table 2: Parameter definition for simulation and integration purposes

2.2.3 Report on the Current Status

As explained in D1.3 (SP-1) a simplified version of the WFS component has been programmed in Matlab in order to carry out the first WFS integration test scheduled for the end of February 2024. Recently, some functionalities regarding blocks 1 and 2 in Table 1 have been updated and integrated in the Matlab script. Moreover, blocks 3 and 4 are now only concerned with asynchronous traffic since block 1 is responsible for isochronous traffic allocation. This Matlab script will be used to build the first step of the API to intercommunicate the WFS with the TSN Wi-Fi node as reported in section 2.2.2. Table 3 shows the designed Matlab functions devoted to the different WFS blocks.

Function	**Input Variables**	**Output Variables**
1 WindowsDesigner_ISO_TIMING	• FlowList • SystemPars	• SuperFrame_ISO
2 DLULResourceAllocation_TIMING	• FlowList • SystemPars	• SlotsDL
3 SchedulerDL_ASYN_TIMING	• FlowList • MS • ServiceClass	• selectedFlowsDL_ASYN • selectedFlowsDL_capacities_ASYN
4 SchedulerUL_ASYN_TIMING	• FlowList • MS • ServiceClass	• selectedFlowsUL_ASYN • selectedFlowsUL_capacities_ASYN
5 FrameBuilder_TIMING	• SlotsDL • SuperFrame_ISO • selectedFlowsDL_ASYN • selectedFlowsDL_capacities_ASYN • selectedFlowsUL_ASYN • selectedFlowsUL_capacities_ASYN • SystemPars • FlowList	• SuperFrame

Table 3: Matlab functions included in the script for first integration test

2.2.4 Report on the stand-alone preliminary tests

The stand-alone preliminary test performed to validate the implementation of the simplified WFS that will be integrated in the first integration test was described in SP-1 D1.3, section 2.3.3. For example, the test was executed in a scenario including 5 DL flows and 3 UL flows (1 packet per flow) of different classes (ISO, ASYN and BE) and were scheduled in one superframe.

The input variables definition was as follows:

```
Nusers=8; % number of users (STAs)
```

```
Nflows=Nusers; % number of flows (we consider 1 flow per user)
Nbuffers=Nflows; % number of buffers
PacketLength=1500; % fixed packet length in bits
BufferSize=1000; % buffer size in packets
SystemPars.NumSlots=73; % number of slot per superframe (DL+UL) (INPUT)
DataCarriers=234; % from 802.11ax (Table 27-79)
OFDMSymbolperSlot=1; % 1 slot = 1 OFDM symbol
SystemPars.ChannelUsesperSlot=DataCarriers*OFDMSymbolperSlot; % channel use =
DataCarriers*OFDMSymbolsperSlot (INPUT)

for n=1:Nbuffers
    buffer_struct(n).BufferSize=BufferSize; %% Number of elements in the
buffer
    buffer_struct(n).LastBufferElement     = 1; %% Position of the last
element in the buffer
    buffer_struct(n).FirstBufferElement    = 1; %% Position of the first
element in the buffer
    buffer_struct(n).NumPackets= 1; %% # of packets in buffer
    buffer_struct(n).PendingBits = PacketLength*buffer_struct(n).NumPackets;
%% Bits pending to be sent (INPUT)
    for jj=1:BufferSize
        buffer_struct(n).Buffer(jj).SizeOfElement = PacketLength; % Number of
bits of a buffer element (packets size)
        buffer_struct(n).Buffer(jj).BitsYetTransmitted = 0;    % Bits
transmitted in previous frames
        buffer_struct(n).Buffer(jj).GenerationTime = 0;        % Packet
arrival time in seconds (INPUT: not used yet)
    end
end

% DL FLOWS
% Class 1 flows (ISO)
FlowList(1).Direction=1; % 1: DL; 2: UL (INPUT)
FlowList(1).ConnexionIdentifier.MS=1; % STA generating/receiving the flow
(INPUT)
FlowList(1).ServiceClassIdentifier=1; % Flow class (INPUT)
FlowList(1).FlowBuffer=buffer_struct(1); % Buffer storing the packets of this
flow
FlowList(1).Periodicity=1; % flow periodicity in superframes (integer number)
(INPUT)
FlowList(1).FirstSlotTXWindow=10; % Slot of the DL frame at which
transmission starts (INPUT)
% Class 2 flows (ASYN)
for n=2:3
    FlowList(n).Direction=1;
    FlowList(n).ConnexionIdentifier.MS=n;
    FlowList(n).ServiceClassIdentifier=2;
    FlowList(n).FlowBuffer=buffer_struct(n);
end
% Class 3 flows (BE)
for n=4:5
    FlowList(n).Direction=1;
```

```
    FlowList(n).ConnexionIdentifier.MS=n;
    FlowList(n).ServiceClassIdentifier=3;
    FlowList(n).FlowBuffer=buffer_struct(n);
end
% UL FLOWS
% Class 1 flows (ISO)
FlowList(6).Direction=2;
FlowList(6).ConnexionIdentifier.MS=6;
FlowList(6).ServiceClassIdentifier=1;
FlowList(6).FlowBuffer=buffer_struct(6);
FlowList(6).Periodicity=1;
FlowList(6).FirstSlotTXWindow=65;
% Class 2 flows (ASYN)
FlowList(7).Direction=2;
FlowList(7).ConnexionIdentifier.MS=7;
FlowList(7).ServiceClassIdentifier=2;
FlowList(7).FlowBuffer=buffer_struct(7);
% Class 3 flows (BE)
FlowList(8).Direction=2;
FlowList(8).ConnexionIdentifier.MS=8;
FlowList(8).ServiceClassIdentifier=3;
FlowList(8).FlowBuffer=buffer_struct(8);

% STA Capacity (bits per channel use = bps/Hz = MCS spectral efficiency)
MS(1).Capacity=6*3/4; % MCS=6 (WiFi6) --> 1053 bits/slot
MS(2).Capacity=2*3/4; % MCS=2 (WiFi6) --> 351 bits/slot
MS(3).Capacity=8*5/6; % MCS=9 (WiFi6)
MS(4).Capacity=1*1/2; % MCS=0 (WiFi6) --> 117 bits/slot
MS(5).Capacity=8*5/6; % MCS=9 (WiFi6) --> 1560 bits/slot
MS(6).Capacity=2*3/4; % MCS=2 (WiFi6)
MS(7).Capacity=2*3/4; % MCS=2 (WiFi6)
MS(8).Capacity=1*1/2; % MCS=0 (WiFi6)

% ISO traffic
% ServiceClass(1).Name='ISO';
ServiceClass(1).ToleratedJitter=1e-6; % TBD (INPUT: not used yet)
ServiceClass(1).MaximumLatency=1e-3; % TBD (INPUT: not used yet)
ServiceClass(1).MaximumPacketErrorRatio=1e-4; % TBD (INPUT: not used yet)
% ASYN traffic
% ServiceClass(2).Name='ASYN';
ServiceClass(2).ToleratedJitter=1e-4;
ServiceClass(2).MinimumRate=0; % TBD (INPUT: not used  yet)
ServiceClass(2).MaximumLatency=1e-3;
ServiceClass(2).MaximumPacketErrorRatio=1e-4;
% BE traffic
% ServiceClass(3).Name='BE';
ServiceClass(3).ToleratedJitter=inf; % TBD (INPUT: not used yet)
ServiceClass(3).MinimumRate=0; % TBD (INPUT: not used yet)
ServiceClass(3).MaximumLatency=inf; % TBD (INPUT: not used yet)
ServiceClass(3).MaximumPacketErrorRatio=1e-3; % TBD (INPUT: not used yet)
```

Obtaining a superframe variable structured as follows:

```
>> SuperFrame.slotsDL = 46

>> SuperFrame.Flow = [ 3   2   2   2   2   2   4   4   4   1   1   4   4   4   4   4   4   4
4   4   4   5   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0   0   0   0   7   7   7   7   7   8   8   8   8   8   8   8   8   8   8   8   8   6   6
6   6   6   0   0   0   0]
```

The length of the SuperFrame.Flow array is the superframe duration in slots. In the example, the superframe is split into 46 slots for the DL (in black) and 27 slots for the UL (in blue). SuperFrame.Flow(3) =2 means that the third slot is assigned to the flow with identifier 2. Isochronous windows are marked in yellow.

2.3 ETHERNET TSN NODES INCLUDING SOFTWARE

2.3.1 Report on the current status

2.3.1.1 The Southbound API between the CNC and the TSN switches

The WRZ-API stands as a specialized interface facilitating communication between the Northbound API and Safran's TSN switches. This API, integrated within the equipment, plays a pivotal role in transmitting network configurations from the CNC to the respective equipment.

We will now delve into the format of the endpoints developed.

2.3.1.1.1 Configuration endpoints

These endpoints are related to the configuration of the TSN network and are categorized into four types: VLAN configuration, TAS configuration, CAM configuration, and QCI configuration. Each of these types encompasses an endpoint through which new configurations can be applied.

Operation	**Description and examples**
/v1/tsn/vlan	
Apply VLAN configuration	Example of Request: {"url": "http://10.22.18.80:8201/v1/tsn/vlan?config_id=5", "method":"POST", "body": "{"wr2": [{"addr": "string", "vlan_id": 0, "vlan_prio": 0, "filter_en": 0, "mac_addr": "string", "proto": 0, "ip": 0, "vlan_id_cfg": 0, "port": 0, "vlan_prio_cfg": 0, "dscp": 0, "dest": 0, "has_dest": 0, "ds": 0, "redundant": 0, "red_dest": 0, "red_handle": 0 }], "wr3": [{...}]}"}
	Response: {"msg": "Success","data": []}

- Notes:
  - **vlan_prio** is an integer between 0-7.
  - The Z16s are divided into 2 quads.
    - For the first quad (wr0-wr3): **dest** and **red_dest** go from 0 to 3.

- For the second quad (wr4-wr7): **dest** and **red_dest** go from 0 to 3.
  - Example: wr4 is 0 (it is the first element from the second quad)

- Values for parameter **ds** are (used in combination with the protocol *port* number):
  - 1: source port
  - 0: destination port
- Values for parameter **redundant** must be 1/0.
- **red_handle** is an integer between 0-16. Must be unique per flow.
- **vlan_id_cfg** and **vlan_prio_cfg** are unused.

/v1/tsn/tas	
Apply TAS configuration	Example of Request: {"url": "http://10.22.18.80:8201/v1/tsn/tas?config_id=5", "method":"POST", "body": "{"wr2": {"rules": [{"interval_time":0, "gate_cfg":0}], "tv_sec": 0, "tv_nsec": 0, "tick_granularity": 0, "framePreemptionStatusTable": 0}, "wr3": {...}}"}
	Response: {"msg": "Success","data": []}

- Notes:
  - **interval_time** is expressed in 16 ns clock cycles. This field would be the result of the following operation: Ceil [T_int (ns) /16].
  - **gate_cfg** is the configuration for each of the TAS queues in one-hot encoding. Example: 0x9 -> 1001 (Q3 and Q0 active).
  - **tv_sec** is the Linux UTC time in which TAS configuration is applied.
  - **tick_granularity** is the size of the step for the interval_time. The default value is 0 which corresponds to 16 ns and it is the recommended value. The other possible values are 1,2,3, and 4, which correspond to 2,3,5 and 9 clock cycles.
  - **framePreemptionStatusTable** indicates the queues that are preemptable in one-hot encoding. Example: 0x8 -> 1000 (Q3 is preemtable). By default, all the queues are express.

/v1/tsn/cam	
Apply CAM configuration	Example of Request: {"url": "http://10.22.18.80:8201/v1/tsn/cam?config_id=5" , "method":"POST", "body": "{"wr2": [{ "vlan_pcp": 0, "vlan_id": 0, "fwd_active": 0, "rx_dest": 0}], "wr3": [{...}]}"}
	Response: {"msg": "Success","data": []}

- Notes:
  - **vlan_pcp** is the priority and it is an integer between 0-7.
  - The Z16s are divided into 2 quads.
    - For the first quad (wr0-wr3): **rx_dest** goes from 0 to 3.
    - For the second quad (wr4-wr7): **rx_dest** goes from 0 to 3.
      - Example: wr4 is 0 (it is the first element from the second quad)

/v1/tsn/qci	
Apply QCI configuration	Example of Request: {"url": "http://10.22.18.80:8201/v1/tsn/qci?config_id=5", "method":"POST", "body": "{"wr2": { "rules": [{"interval_time": 0, "gate_cfg": 0}], "tv_sec": 0, "tv_nsec": 0, "qci_tick_granularity": 0}, "wr3": {...}}"}
	Response: {"msg": "Success","data": []}

- Notes:
  - **interval_time** is expressed in 16 ns clock cycles. This field would be the result of the following operation: Ceil [T_int (ns) /16].
  - **gate_cfg** is the configuration for each of the QCI queues in one-hot encoding. Example: 0x9 -> 1001 (Q3 and Q0 active).
  - **tv_sec** is the Linux UTC time in which QCI configuration is applied.
  - **qci_tick_granularity** is the size of the step for the interval_time. The default value is 0 which corresponds to 16 ns and it is the recommended value. The other possible values are 1,2,3, and 4, which correspond to 2,3,5 and 9 clock cycles.

Possible error responses	
422: Unprocessable Entity	{ "detail": [{ "loc": ["body", "wr5", "tv_sec"],"msg": "value is not a valid integer", "type": "type_error.integer" }]}
	{"detail":[{"loc":["body",166],"msg": "Expecting ',' delimiter: line 12 column 5 (char 166)","type":"value_error.jsondecode","ctx": {"msg": "Expecting ',' delimiter", "doc": "{\n \"wr9\": {\n \"rules\": [\n {\n \"interval_time\": 0,\n \"gate_cfg\": 0\n }\n],\n \"tv_sec\": 3,\n \"tv_nsec\": 0,\n \"tick_granularity\": 0\n \"framePreemptionStatusTable\": 0\n }\n}", "pos": 166, "lineno": 12, "colno": 5}}]}
	{"detail": [{ "loc": ["body", "wr2","tv_sec"],"msg": "field required","type": "value_error.missing"}]}

2.3.1.1.2 Generating reports endpoints

These endpoints aim to generate data about packet traffic within the Z16.

Operation	Description and examples
/v1/tsn/reports	
Get Reports	Example of Request: {"url": "http://10.22.18.80:8201/v1/tsn/reports", "method":"GET"}
	Response: {"msg": "Success", "data": ["/media/data/tsn/reports/22-dic_iface_1"]}

Start Report	Example of Request: {"url": "http://10.22.18.80:8201/v1/tsn/reports?iface_num=1&vid=1&pcp=0&seq_off=22&seq_mask=ffff&length=16&frer=0&file_name=prueba&time_interval=60", "method":"POST"}
	Response: {"msg": "Success","data": []}

- Notes:
  - **frer** must be 1/0 and it is the flag to compensate the offset value to get the sequence number if redundancy headers are present.

Download Report	Example of Request: {"url": "http://10.22.18.80:8201/v1/tsn/reports/prueba", "method": "GET"}
	Example of Response: ----------New Capture Session with VID(1) PCP(0) Port (1)---------- TS UTC (s),TS (ns),Sequence Number 1703241856,346389712,326 1703241856,446535056,327 1703241918,34762608,942 1703241918,134901904,943

Possible error responses	

404: Not Found	{
	"msg": "File 'report_1' not found",
	"data": []
	}

2.3.1.2 The Northbound API between the CNC and the TSN Controller

The Northbound API between the CNC and the TSN Controller implements a generic REST API for receiving the optimized operational settings calculated at UPC's TSN Controller.

2.3.1.2.1 Network equipment management

These are the endpoints responsible for managing the settings of the Z16 nodes stored within the CNC's database.

Operation	Description and examples
/v1/device	
Get Devices	Example of Request: {"url": "http://localhost:8201/v1/devices", "method": "GET"} or {"url": "http://localhost:8201/v1/devices?name=new_device", "method": "GET"}

	Example of Response:
	[{
	"scrapping_data": false,
	"id": 1,
	"host": "10.22.18.12",
	"port": 8201,
	"active": true,
	"name": "12",
	"sync_mode": null
	}
	]

Create Device	Example of Request: {"url": "http://localhost:8201/v1/device, "method":"POST", "body" : "{"name": "string", "host": "198.51.100.42", "active": false, "scrapping_data": false, "sync_mode": "string", "port": 8201}"}
Modify Device	Example of Request: {"url": "http://localhost:8201/v1/device?name=new_device", "method": "PATCH", "body": "{"name": "string", "host": "198.51.100.42", "active": false, "scrapping_data": false, "sync_mode": "string", "port": 0}"}
Delete Device	Example of Request: {"url": "http://localhost:8201/v1/device?name=new_device", "method": "PATCH"}

Possible error responses	

404: Not Found	{
	"detail": "Device not found"
	
	}

422: Unprocessable Entity	{
	"detail": "active: Input should be a valid boolean, unable to interpret input"
	}

	{
	"detail": "sync_mode: Input should be a valid string"
	}

	{
	"detail": "host: Input is not a valid IPv4 address"
	}

	{
	"detail": "name: Field required"
	}

409: Conflict	{
	"detail": "Host already registered"
	}

	{
	"detail": "Name already registered"
	}

2.3.1.2.2 Network configuration

These endpoints are used to configure the TSN network.

VLAN

Operation	Description and examples
/v1/net-config/vlan	
Get VLAN Configuration	Example of Request: {"url": "http://localhost:8201/v1/net-config/vlan", "method": "GET"} or {"url": "http://localhost:8201/v1/net-config/vlan?device=new_device&iface=wr2", "method": "GET"} or {"url": "http://localhost:8201/v1/net-config/vlan?device=new_device", "method": "GET"}
Apply VLAN Configuration	Example of Request: {"url": "http://localhost:8201/v1/net-config/vlan, "method":"POST", "body" : "[{"device": "string", "iface": "wr2", "rules": [1,3]}]"}
Delete VLAN Configuration	Example of Request: {"url": "http://localhost:8201/v1/net-config/vlan?device=new_device&iface=wr2", "method": "DELETE"}

GET VLAN Rules	Example of Request: {“url”: “http://localhost:8201/v1/net-config/vlan/rule”, “method”: “GET”} Or {“url”: “http://localhost:8201/v1/net-config/vlan/rule?id=3”, “method”: “GET”}
Create VLAN Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/vlan/rule”, “method”: “POST”, “body”: “{“addr”: “string”, “vlan_id”: 0, “vlan_prio”: 0, “filter_en”: 0, “mac_addr”: “string”, “proto”: 0, “ip”: 0, “vlan_id_cfg”: 0, “port”: 0, “vlan_prio_cfg”: 0, “dscp”: 0, “dest”: 0, “has_dest”: 0, “ds”: 0, “redundant”: 0, “red_dest”: 0, “red_handle”: 0}”}
Modify VLAN Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/vlan/rule?id=3”, “method”: “PATCH”, “body”: “{“addr”: “string”, “vlan_id”: 0, “vlan_prio”: 0, “filter_en”: 0, “mac_addr”: “string”, “proto”: 0, “ip”: 0, “vlan_id_cfg”: 0, “port”: 0, “vlan_prio_cfg”: 0, “dscp”: 0, “dest”: 0, “has_dest”: 0, “ds”: 0, “redundant”: 0, “red_dest”: 0, “red_handle”: 0}”}
Delete VLAN Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/vlan/rule?id=3”, “method”: “DELETE”}

- Notes:
  - **vlan_id_cfg** and **vlan_prio_cfg** are unused.

TAS

Operation	**Description and examples**
/v1/net-config/tas	
Get TAS Configuration	Example of Request: {“url”: “http://localhost:8201/v1/net-config/tas”, “method”: “GET”} or {“url”: “http://localhost:8201/v1/net-config/tas?device=new_device&iface=wr2”, “method”: “GET”} or {“url”: “http://localhost:8201/v1/net-config/tas?device=new_device”, “method”: “GET”}
Apply TAS Configuration	Example of Request: {"url": “http://localhost:8201/v1/net-config/tas, "method":"POST", “body” : “[{“device”: “string”, “iface”: “wr2”, “rules”: [1,3], “tv_sec”: 0, “tv_nsec”: 0, “tick_granularity”: 0, “framePreemptionStatusTable”: 0}]”}
Delete TAS Configuration	Example of Request: {“url”: “http://localhost:8201/v1/net-config/tas?device=new_device&iface=wr2”, “method”: “DELETE”}
GET TAS Rules	Example of Request: {“url”: “http://localhost:8201/v1/net-config/tas/rule”, “method”: “GET”}

	Or {“url”: “http://localhost:8201/v1/net-config/tas/rule?id=3”, “method”: “GET”}
Create TAS Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/tas/rule”, “method”: “POST”, “body”: “{“interval_time”: 0, “gate_cfg”: 0}”}
Modify TAS Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/tas/rule?id=3”, “method”: “PATCH”, “body”: “{“interval_time”: 0, “gate_cfg”: 0}”}
Delete TAS Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/tas/rule?id=3”, “method”: “DELETE”}

CAM

Operation	**Description and examples**
/v1/net-config/cam	
Get CAM Configuration	Example of Request: {“url”: “http://localhost:8201/v1/net-config/cam”, “method”: “GET”} or {“url”: “http://localhost:8201/v1/net-config/cam?device=new_device&iface=wr2”, “method”: “GET”} or {“url”: “http://localhost:8201/v1/net-config/cam?device=new_device”, “method”: “GET”}
Apply CAM Configuration	Example of Request: {"url": “http://localhost:8201/v1/net-config/cam, "method":"POST", “body” : “[{“device”: “string”, “iface”: “wr2”, “rules”: [1,3]}]”}
Delete CAM Configuration	Example of Request: {“url”: “http://localhost:8201/v1/net-config/cam?device=new_device&iface=wr2”, “method”: “DELETE”}
GET CAM Rules	Example of Request: {“url”: “http://localhost:8201/v1/net-config/cam/rule”, “method”: “GET”} or {“url”: “http://localhost:8201/v1/net-config/cam/rule?id=3”, “method”: “GET”}
Create CAM Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/cam/rule”, “method”: “POST”, “body”: “{“vlan_pcp”: 0, “vlan_id”: 0, “fwd_active”: 0, “rx_dest”: 0}”}
Modify CAM Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/cam/rule?id=3”, “method”: “PATCH”, “body”: “{“vlan_pcp”: 0, “vlan_id”: 0, “fwd_active”: 0, “rx_dest”: 0}”}

Delete CAM Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/cam/rule?id=3”, “method”: “DELETE”}

QCI

/v1/net-config/qci	
Get QCI Configuration	Example of Request: {“url”: “http://localhost:8201/v1/net-config/qci”, “method”: “GET”} or {“url”: “http://localhost:8201/v1/net-config/qci?device=new_device&iface=wr2”, “method”: “GET”} or {“url”: “http://localhost:8201/v1/net-config/qci?device=new_device”, “method”: “GET”}
Apply QCI Configuration	Example of Request: {"url": “http://localhost:8201/v1/net-config/qci, "method":"POST", “body” : “[{“device”: “string”, “iface”: “wr2”, “rules”: [1,3], “tv_sec”: 0, “tv_nsec”: 0, “qci_granularity”: 0}]”}
Delete QCI Configuration	Example of Request: {“url”: “http://localhost:8201/v1/net-config/qci?device=new_device&iface=wr2”, “method”: “DELETE”}
GET QCI Rules	Example of Request: {“url”: “http://localhost:8201/v1/net-config/qci/rule”, “method”: “GET”} or {“url”: “http://localhost:8201/v1/net-config/qci/rule?id=3”, “method”: “GET”}
Create QCI Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/qci/rule”, “method”: “POST”, “body”: “{“interval_time”: 0, “gate_cfg”: 0}”}
Modify QCI Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/qci/rule?id=3”, “method”: “PATCH”, “body”: “{“interval_time”: 0, “gate_cfg”: 0}”}
Delete QCI Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/qci/rule?id=3”, “method”: “DELETE”}

2.3.1.2.3 Latency and packet loss test generation

These endpoints are used to generate reports on packet traffic in the TSN network.

/v1/net-analyzer	
Start Report	Example of Request: {“url”: “http://localhost:8201/v1/net-analyzer?vid=1&pcp=0&seq_offset=22&mask=ffff&seq_length=16”,

	“method”: “POST”, “body”: “{“name”: “test”, “duration”: 1, “devices”: [{“name”: “new_device”, “iface”: 2, “frer”: 0}]}”}

	Example of Response:
	{
	"date": "2023-12-27T18:51:52.885682",
	"id": 42,
	"name": "test",
	"status": "In Progress",
	"duration": 60
	}

GET Reports	Example of Request: {"url": “http://localhost:8201/v1/net-analyzer”, "method":"GET"} Or {"url": “http://localhost:8201/v1/net-analyzer?name=test”, "method":"GET"}

	Example of Response:
	[
	{
	"date": "2023-10-25T17:51:45.631055",
	"id": 1,
	"name": "test",
	"status": "Success",
	"duration": 60
	}
	]

Delete Report	Example of Request: {“url”: “http://localhost:8201/v1/net-analyzer?name=test”, “method”: “DELETE”}
Download Report	Example of Request: {“url”: “http://localhost:8201/v1/net-analyzer/downloader?name=test”, “method”: “GET”}
Create QCI Rule	Example of Request: {“url”: “http://localhost:8201/v1/net-config/qci/rule”, “method”: “POST”, “body”: “{“interval_time”: 0, “gate_cfg”: 0}”}
GET latency report	Example of Request: {“url”: “http://localhost:8201/v1/net-analyzer/latency?name=test”, “method”: “GET”}
GET packets lost report	Example of Request: {“url”: “http://localhost:8201/v1/net-analyzer/packets-lost?name=test”, “method”: “GET”}

- Notes:
  - **frer** must be 1/0 and it is the flag to compensate the offset value to get the sequence number if redundancy headers are present.

Possible error responses	

422: Unprocessable Entity	{
	"detail": "Duration should be an integer between 1 and 120"
	}

	{
	"detail": "duration: Input should be a valid integer, unable to parse string as an integer"
	}

	{
	"detail": "Iface from 10.22.18.10 should be an integer between 0 and 6"
	}

	{
	"detail": "duration: Field required"
	}

	{
	"detail": "Length should be 8/16/24/28"
	}

	{
	"detail": "Sequence offset should be an integer bigger than 14"
	}

	{
	"detail": "Priority should be an integer between 0 and 7"
	}

409: Conflict	{
	"detail": "Device string does not exist"
	}

	{
	"detail": "Report already exists"
	}

	{
	"detail": "Device 10.22.18.12 is not active"
	}

400: Bad Request	{
	"detail": "Cannot make request: http://10.22.18.10:8201/v1/tsn/reports"
	}

404: Not Found	{
	"detail": "Report not found"
	}

2.3.1.2.4 Network topology visualization

This endpoint returns a dictionary containing information about the connections of the devices.

/v1/net-analyzer/topology	
GET Topology	Example of Request: {“url”: “http://localhost:8201/v1/net-analyzer/topology”, “method”: “GET”}
	Example of Response: { "10.22.18.12": [], "10.22.18.9": [{ "iface": "wr1", "system_name": "z16-401", "address": "10.22.18.10", "desc": "wr0" }]}

2.3.1.2.5 Export Metrics Settings

This endpoint allows for obtaining a list of monitored devices, as well as adding or removing devices.

/v1/export-metrics-settings	
SET export-metrics variables	Example of Request: {“url”: “http://localhost:8201/v1/export-metrics-settings?export_metrics=no”, “method”: “POST”} or {“url”: “http://localhost:8201/v1/export-metrics-settings?export_metrics=yes&cloud_org_id=string&cloud_tsdb_bucket=string&cloud_tsdb_token=string”, “method”: “POST”}
GET devices monitorized	Example of Request: {“url”: “http://localhost:8201/v1/export-metrics-settings/targets”, “method”: “GET”}
	Example of Response: ["10.22.18.10", "10.22.18.12", "10.22.18.13", "10.22.18.80", "10.22.18.81", "10.22.18.9"]

Modify devices monitorized	Example of Request: {“url”: “http://localhost:8201/v1/export-metrics-settings/targets?job=wrz”, “method”: “POST”, “body”: “[“198.51.100.42”, “10.22.18.80”]”}
Reload	Example of Request: {“url”: “http://localhost:8201/v1/export-metrics-settings/reload-config”, “method”: “POST”}

Possible error responses	

404: Not Found	{
	"detail": "Job wrz2 not found"
	}

422: Unprocessable Entity	{
	"detail": "export_metrics must be 'yes' or 'no'"
	}

	{
	"detail": "cloud_tsdb_bucket cannot be empty"
	}

	{
	"detail": "cloud_org_id cannot be empty"
	}

	{
	"detail": "cloud_tsdb_token cannot be empty"
	}

2.3.1.3 Workflow

In this section we will explain the workflow designed between the TSN Controller, CNC, and the Z16 devices for monitoring and configuring the TSN network. Figure 2-6 illustrates the flow of information exchange among these three components. The downward arrows symbolize the process of network configuration, initiated by the TSN Controller, translated through the CNC API to the WRZ API, and subsequently distributed to the respective devices via the WRZ API. Conversely, the upward arrows signify the network monitoring process, where information is extracted from the devices using the SNMP Exporter, managed by the CNC, and eventually relayed to the TSN Controller.

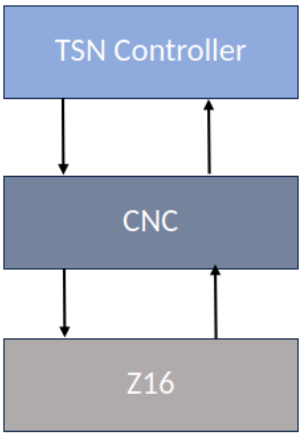


Figure 2-6: Ethernet TSN workflow

Configure the TSN Network

The steps to be taken to configure the TSN Network are (downward arrows):

Preliminary steps (0.1 and 0.2):

0.1. The CNC API is used to add the devices that are on the TSN network.

Example:

Request	

POST /v1/device	{
	"name": "new_device",
	"host": "10.22.18.80",
	"active": false,
	"scrapping_data": false,
	"sync_mode": "string",
	"port": 8201
	}

0.2. Configuration rules are created using the CNC

Example – Create TAS rule:

If we want the following GCL specification:

- Slot #0: Duration 1 ms, Gate Settings: Q3 & Q0 open.

The parameters we should put are:

- **interval_time:** result of the operation Ceil [T_int (ns) /16]

    - In this case, Ceil [1000000 / 16] = 62500
- **gate_cfg**: Q3 and Q0 open is 1001 -> 0x9

The request will be:

Request	

POST /v1/net-config/tas/rule	{
	"interval_time": 62500,
	"gate_cfg": 9
	}

Response	

{
"interval_time": 62500,
"gate_cfg": 9,
“id”: 1
}	

The rule created has ***id = 1***.

Example - Create VLAN rule:

If we want the following VLAN configuration:

- WR0: VID: 1; PCP: 2; MAC_DST: 0xcafebabe0102; MAC_ADDR: 0xaabbccddeeff; HAS_DEST:1; DEST:1.
- WR1: VID: 1; PCP: 2; MAC_DST: 0xcafebabe0102; MAC_ADDR: 0xaabbccddeeff.

The parameters we should put are:

- wr0:
    - "addr": "0xcafebabe0102"
    - "vlan_id": 1,
    - "vlan_prio": 2,
    - "mac_addr": "0xaabbccddeeff",
    - "dest": 1,
    - "has_dest": 1
- wr1:
    - "addr": "0xcafebabe0102"
    - "vlan_id": 1,
    - "vlan_prio": 2,
    - "mac_addr": "0xaabbccddeeff",

The requests will be:

- Rule for wr0

Request	
POST /v1/net-config/vlan/rule	{"addr": "0xcafebabe0102","vlan_id": 1, “vlan_prio": 2, "filter_en": 0,"mac_addr": "0xaabbccddeeff ","proto": 0,"ip": 0,"vlan_id_cfg": 0,"port": 0,"vlan_prio_cfg": 0, "dscp": 0, "dest": 1,"has_dest": 1, "ds": 0, "redundant": 0,"red_dest": 0, "red_handle": 0}
Response	
{“id”:1, "vlan_id":1, "vlan_prio":2, "filter_en":0, "mac_addr":"0xaabbccddeeff", "proto":0, "ip":0, "vlan_id_cfg":0, "port": 0,"vlan_prio_cfg": 0, "dscp": 0,"dest": 1,"has_dest": 1,"ds": 0,"redundant": 0,"red_dest": 0,"red_handle": 0}	

- Rule for wr1

Request	
POST /v1/net-config/vlan/rule	{"addr": "0xcafebabe0102","vlan_id": 1, “vlan_prio": 2, "filter_en": 0,"mac_addr": "0xaabbccddeeff ","proto": 0,"ip": 0,"vlan_id_cfg": 0,"port": 0,"vlan_prio_cfg": 0, "dscp": 0, "dest": 0,"has_dest": 0, "ds": 0, "redundant": 0,"red_dest": 0, "red_handle": 0}
Response	
{“id”:2, "vlan_id":1, "vlan_prio":2, "filter_en":0, "mac_addr":"0xaabbccddeeff", "proto":0, "ip":0, "vlan_id_cfg":0, "port": 0,"vlan_prio_cfg": 0, "dscp": 0,"dest": 0,"has_dest": 0,"ds": 0,"redundant": 0,"red_dest": 0,"red_handle": 0}	

wr0’s rule has **id = 1** and wr1’s rule has **id = 2**.

1. TSN Controller decides which network configuration is going to be used and the corresponding rules are applied with the CNC API.

Example – Apply TAS rule:

In rules we put the ***id*** of the rule we have previously created.

Request	
POST /v1/net-config/tas	[{ "device": “new_device”, "rules": [1],"tv_sec": 1703238797,"tv_nsec": 0, "tick_granularity": 0,"framePreemptionStatusTable": 0 }]

Parameter **tv_sec** is the Linux UTC time in which TAS configuration is applied and should be calculated in the following way:

- <Current Network Epoch> + Additional offset to make up for configuration time.

Example – Apply VLAN rule:

In rules we put the ***id*** of the rule we have previously created.

Request	
POST /v1/net-config/vlan	[{ "device": "new_device", "iface": "0", "rules": [1] }]

Request	
POST /v1/net-config/vlan	[{ "device": "new_device", "iface": "1", "rules": [2] }]

The following steps are done automatically, without user control.

2. The CNC API makes a call to the WRZ API of the corresponding device to apply the network configuration.

3. WRZ API applies the chosen configuration to the devices.

Monitor the TSN Network

To monitor the TSN Network the workflow is:

1. SNMP collects the metrics defined in the CNC.

2. CNC saves the collected metrics and exports them to an external database so that TSN Controller can access them.

3. TSN Controller analyses the collected data and, based on it, decides which network configuration to apply.

2.4 SDN-TSN CONTROLLER

2.4.1 Description

The Software Defined Networking (SDN)- Time-Sensitive Networking (TSN) controller (SDN-TSN) is the entity responsible for the configuration of connectivity services within the TSN-based network infrastructures, and the corresponding management of said infrastructure. To do so, it engages on one hand with the Connectivity Manager (CM), which is the entity that receives service provisioning requests, which ultimately translate to connectivity requests towards the SDN-TSN controller; and, on the other hand, it engages with the underlying TSN-capable network nodes (WiFi and Ethernet) at the data plane, to configure them according to the

characteristics of the requested connectivity. Figure 2-7 depicts a schematic of the functional architecture of the module and the interactions with other modules, namely, the CM and the TSN-enabled data plane, as well as the main data structures treated within each of the individual functionalities. The communications are achieved thanks to a pair of interfaces: the Northbound Interface (NBI) and the Southbound Interface (SBI).

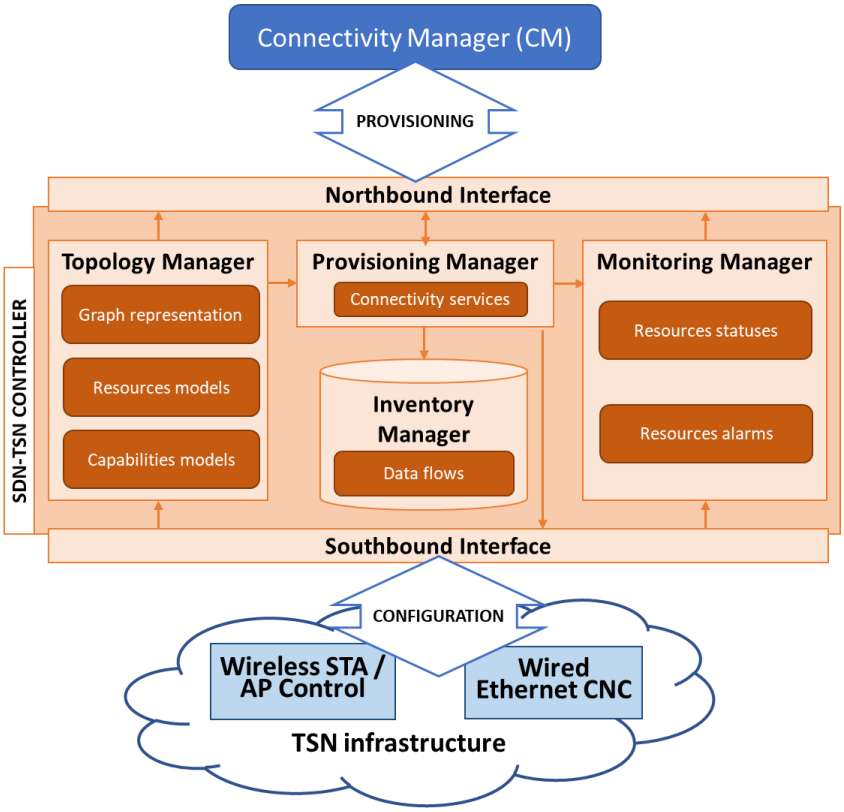


Figure 2-7: SDN-TSN Controller architecture.

The core of the functionalities is as listed below:

- Topology Manager (TM): responsible for keeping a graph representation of the underlying data plane, the network resources and their capabilities. It extracts this information via the SBI.
- Inventory Manager (IM): contains the inventory of the established data flows, according to the outputs of the connectivity provisioning operations performed at the Provisioning Manager. It exposes this information to the CM via the Provisioning Manager through the NBI.
- Provisioning Manager (PM): centralizes the provisioning of data flows to satisfy the requirements of connectivity services coming from the NBI. The low level configurations are decided thanks to the information stored at the Topology and Inventory Managers, and executes the computed configuration by means of the SBI.
- Monitoring Manager (MM): it collects the current status of the resources at the TSN-capable data plane as well as potential alarms via the SBI. The collected information is exposed to the CM thanks to the NBI.

2.4.2 Report on the Current Status

The advancements made for the current version of the SDN-TSN are mainly in two fronts: on one hand, the core functionalities of both the provisioning and the topology managers have been further developed to include the main classes, data structures and logic for their operation; on the other hand, both NBI and SBI have been expanded and validated for the operations of flow creation and deletion according to connectivity services requests coming from the CM (NBI) and the de-configuration of data plane nodes as a response of a flow deletion (SBI).

Starting with the NBI, Figure 2-8 depicts a schematic of the data structure of the flow of an application/service, and the relationship between sub-structures. Using such structure, the operations for the creation and deletion of flows have been defined. Table 4 and Table 5 provide the details of the operations and the parameters of the requests.

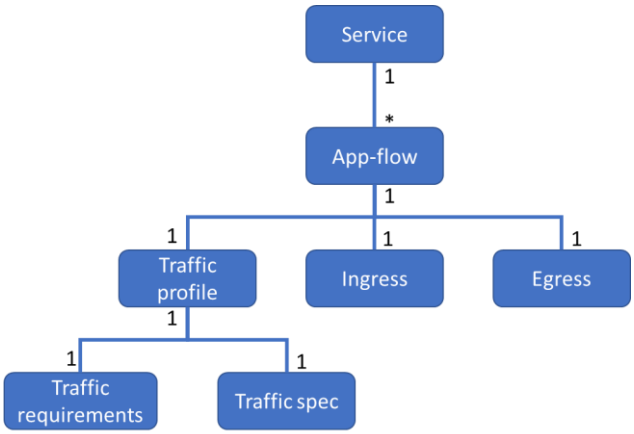


Figure 2-8: Data structure tree of a service request.

Section	Northbound Interface
Type	Service's flow creation
TIMING Target objective	Create the data flows related to upper layer service requests according to the state connectivity and traffic requirements
Path	**POST /flows** HTTPS/1.1 *Description: Creates a TSN or best effort flow.*
Body	Example of schema:

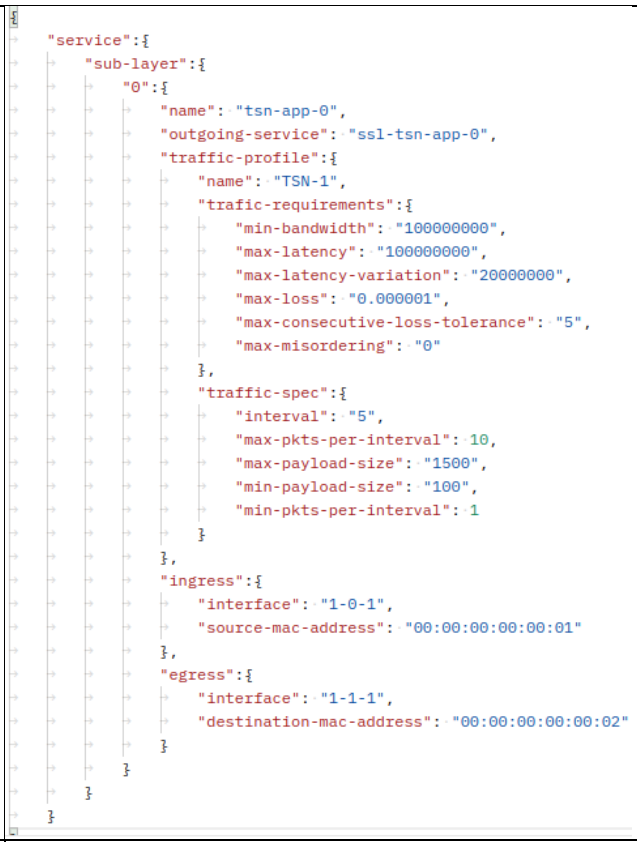
	
Response	It return a string representation of the created flow identifier (if successful) and the response code as below: 200 (OK): creation operation is successful 400 (Bad request): the request body is malformed or some field is missing 404 (Not found): the creation request cannot be fulfilled (e.g., due to the lack of network resources) 500 (Internal Server Error): Unspecified error during the flow creation

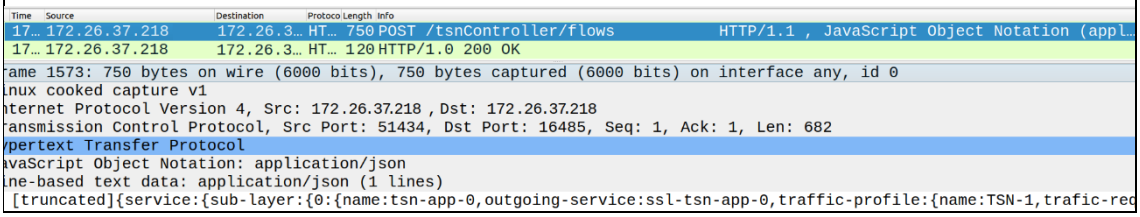
Table 4: Description of the operation for creating a data flow

Section	Northbound Interface
Type	Service's flow deletion
TIMING Target objective	Delete the active data flows as part of the decommissioning of an upper layer service
Path	**DELETE /flows/{flowID}** HTTPS/1.1 *Description: Deletes the flow with identifier flowID*
Body	None
Response	No information is returned in the body of the response. Only the response code is returned: 200 (OK): delete operation is successful 404 (Not found): no flow with the specified identifier is currently active 500 (Internal Server Error): Unspecified error during the flow deletion

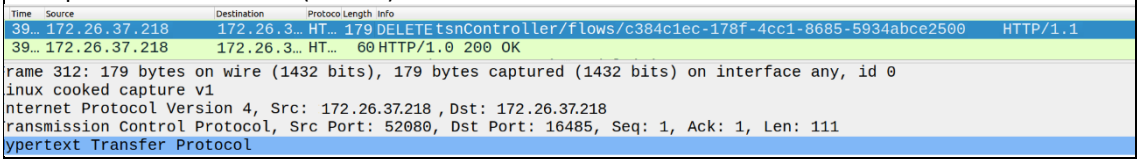
Table 5: Description of the operation for deleting a data flow

The remote access to the presented NBI operations has been validated with the tests reported below. The current tests expand what have been reported already in deliverable SP1 - D1.3, which focused on the flow and topology status dissemination. In addition, let us mention that the time from flow creation/deletion request to data plane configuration/de-configuration operation, that is, the time that the SDN-TSN requires for processing upper layer requests, transform them into specific configurations and issue them towards the data plane via the SBI, has been measured and is in the order of several tens to few hundreds of milliseconds, depending on the current load of the server in which the SDN-TSN controller is running. With this being said, the tests are reported below, for which we showcase Wireshark captures of the successful exchanges between a test REST client and the NBI of the controller.

Test #001	
Test Id	TSN-C_NBI_createFlow
Description	
Test related to the Northbound Interface (NBI) of the SDN-TSN controller to verify the correct reception and processing of data flow creation requests.	
Expected Results	
The NBI accepts the operation and processes the body of the request.	
Output	
The operation is successful (200 OK)	



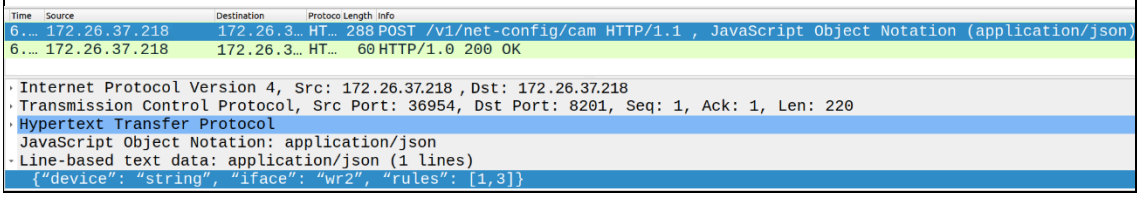
Test #002	
Test Id	TSN-C_NBI_deleteFlow
Description	
Test related to the Northbound Interface (NBI) of the SDN-TSN controller to verify the correct reception and processing of data deletion creation requests.	
Expected Results	
The NBI accepts the operation and processes the body of the request.	
Output	
The operation is successful (200 OK)	



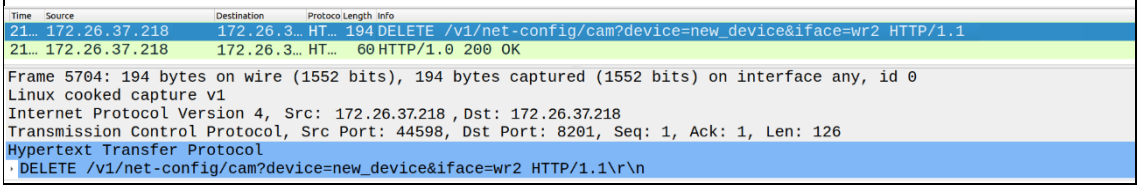
In regards to the SBI, the SDN-TSN Controller implements the API reported in Section 2.3.1.2 of this same document. For the sake of conciseness, we refer to the description of the operations and parameters already described. The SBI has been further developed to consider an expanded set of operation with respect what was previously reported in SP1 - D1.3, essentially, the configuration of the CAM as well as the deletion of VLAN, TAS and CAM configurations. In below

we report several experiments that validate the implemented operations. To this end, a mock-up REST server emulating the NBI of CNC has been employed so as to validate that the SDN-TSN Controller issues properly the operation, thus, the SBI works as expected.

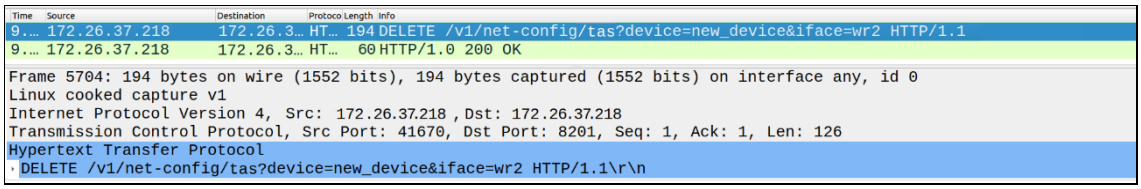
Test #003	
Test Id	TSN-C_SBI_configureCAM
Description	
Test related to the SBI of the SDN-TSN Controller to validate the sending of configurations for the CAM	
Expected Results	
The SBI sends a well-formatted operation and is accepted by the mock-up REST server	
Output	
The operation is successful (200 OK)	



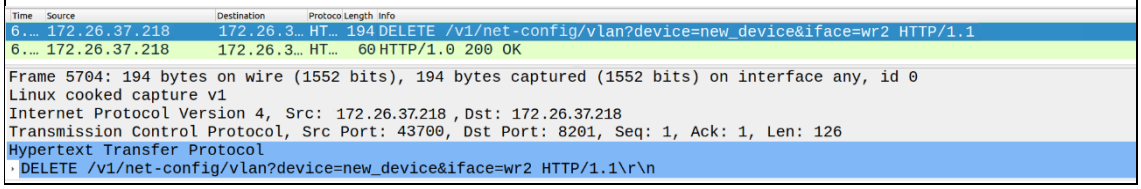
Test #004	
Test Id	TSN-C_SBI_deleteCAM
Description	
Test related to the SBI of the SDN-TSN Controller to validate the sending of delete requests for CAM configurations	
Expected Results	
The SBI sends a well-formatted operation and is accepted by the mock-up REST server	
Output	
The operation is successful (200 OK)	



Test #005	
Test Id	TSN-C_SBI_configureTAS
Description	
Test related to the SBI of the SDN-TSN Controller to validate the sending of delete requests for TAS configurations	
Expected Results	
The SBI sends a well-formatted operation and is accepted by the mock-up REST server	
Output	
The operation is successful (200 OK)	



Test #006	
Test Id	TSN-C_SBI_configureVLAN
Description	
Test related to the SBI of the SDN-TSN Controller to validate the sending of delete requests for VLAN configurations	
Expected Results	
The SBI sends a well-formatted operation and is accepted by the mock-up REST server	
Output	
The operation is successful (200 OK)	



2.5 CONNECTIVITY MANAGER

2.5.1 Description

The Time-Sensitive Networking (TSN) Connectivity Manager (CM) is a key component in the TIMING architecture, playing as the cornerstone role in managing and orchestrating time-sensitive networks. In the context of the TIMING project, the TSN CM leverages the rest of the TIMING control plane component to ensure efficient, reliable, and timely data transport across the network. The TSN CM is strategically positioned at the top of the TIMING architecture as depicted in Figure 2-9. It is designed to interact directly with all TSN and Metro SDN (Software-Defined Networking) Controllers, as well as the Digital Twin (DT). This central placement allows the TSN CM to have an overarching view of the network, enabling it to perform its functions effectively.

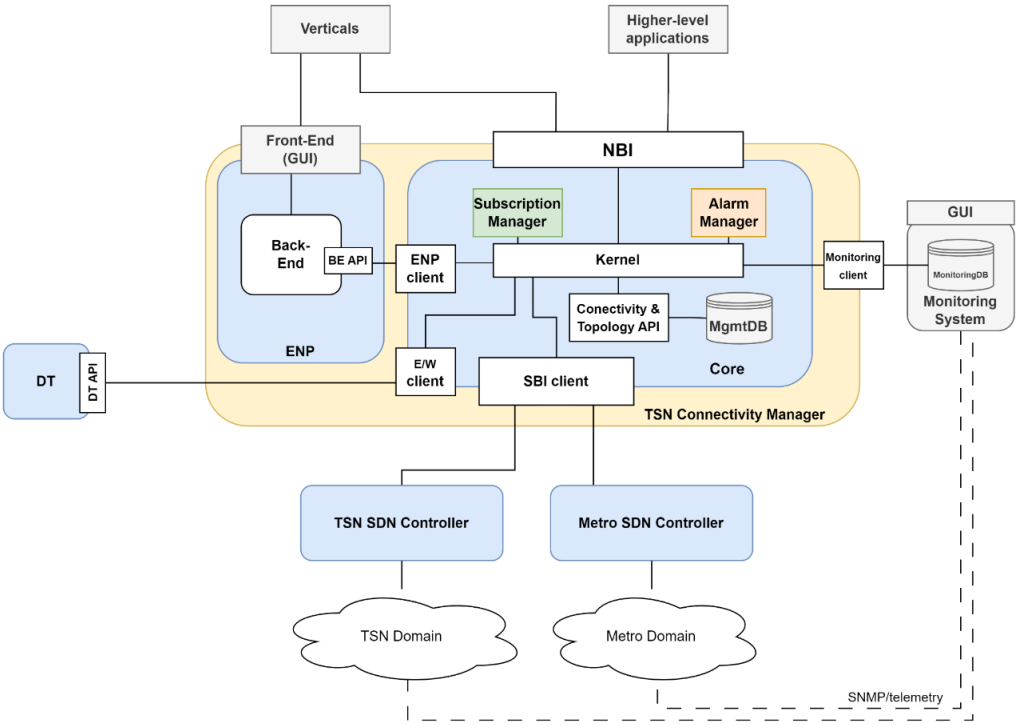


Figure 2-9: TSN Connectivity Manager Architecture

To fulfil its role, the TSN connectivity manager does so by leveraging a set of functionalities tailored to manage and optimize time-sensitive data flows across the network, which are summarized below.

- *Network Resource Discovery and Management:* One of the key functionalities of the TSN CM is to discover the resources and capacities available within the TSN network. This involves identifying available bandwidth, latency characteristics, and other critical network parameters.
- *Configuration and Optimization:* Post discovery, the TSN CM summarizes the main changes needed in the network and communicates these to the underlying controllers. This process involves deploying new configurations to optimize network performance, especially for time-sensitive data flows.
- *Integration with Digital Twin for Predictive Analysis:* The TSN CM collaborates with the Digital Twin (DT) to simulate network conditions and predict the performance of various network configurations. This predictive modeling is essential for making informed decisions about network configuration changes.
- *Optimization Algorithms:* Utilizing advanced algorithms such as the Dijkstra algorithm, the TSN CM efficiently manages the routing of data, ensuring the most optimal paths are chosen for time-sensitive traffic.

The TSN CM utilizes a range of interfaces to interact with different components of the TIMING architecture. The TSN CM makes use of three main APIs: Southbound Interface (SBI), Northbound Interface (NBI), and East/West Interface (E/WBI).

- *SouthBound Interface (SBI)*: This interface is used for reading underlying topological information, which is crucial for understanding the current state of the network.
- *NorthBound Interface (NBI)*: It facilitates interactions with other TIMING modules and higher-level applications. This interface is vital for global and automated management of the architecture.
- *East/WestBound Interface (E/WBI)*: This interface enables the TSN CM to communicate with the Digital Twin, sending and receiving information necessary for simulating network conditions.

2.5.2 Report on the Current Status

The overall progress of the development of the TSN Connectivity Manager is on-track and with solid advances, specially, in the definition of internal models and in the Core and ENP functional blocks, as presented in D1.4 (SP1) and in this document, see later sections.

For future steps, it is planned to develop the external client-side modules (SBI and E/WBI) to allow the expected interaction with the controllers (via SBI) and with the Digital Twin (E/WBI). In addition, integration tests will be performed between all components involved. Potentially, as a consequence of the integration phase, some refinements to the TSN CM data model may be carried out.

2.5.3 NorthBound Interface Implementation

One of the major advances of the TSN Connectivity Manager has been on the definition and implementation of the NorthBound interface, that allows vertical and upper-layer applications to communicate with it. The revised architecture of the Northbound Interface (NBI) adopts a modular design, partitioning various functions into separate, distinct endpoints. As depicted in **Error! Reference source not found.**, this architecture is comprised of three principal categories: controllers, flows, and topologies. Each category is engineered to fulfill a specific role within the broader context of network management, ensuring a more streamlined and efficient operational framework.

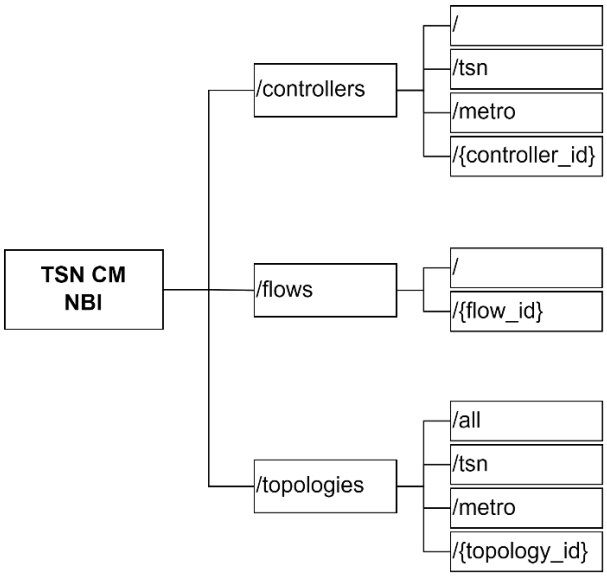


Figure 2-10: TSN Connectivity Manager NBI endpoints structure

Thus, the current version of the TSN CM NBI is implemented as follows:

<table>
<tr><th colspan="2">/controllers</th></tr>
<tr><th>Operation</th><th>Description and examples</th></tr>
<tr><td>POST controller</td><td>It registers a controller in the TSN Connectivity Manger context.

Request URL: POST https://{CM-IP}:8003/controllers/

Example schemas (JSON-based):

INPUT:
<pre>{
  "name": "default_name",
  "description": "default_description",
  "url": "https://localhost",
  "port": 0,
  "username": "default_username",
  "password": "default_password",
  "type": "type_not_defined"
}</pre>
OUTPUT:

<pre>{
  "controller_id": "string",
  "name": "string",
  "description": "string",
  "url": "string",
  "port": 0,
  "username": "string",
  "password": "string",
  "type": "string"
}</pre></td></tr>
<tr><td>GET controllers</td><td>Returns a list with the information of all registered controllers in the TSN Connectivity Manager

Request URL: GET https://{CM-IP}:8003/controllers/

Example schemas (JSON-based):

INPUT:
-

OUTPUT:

<pre>[
  {
    "controller_id": "string",
    "name": "string",
    "description": "string",
    "url": "string",
    "port": 0,
    "username": "string",
    "password": "string",</pre></td></tr>
</table>

	"type": "string" }]

DELETE controllers	Delete from the TSN CM system all the registered controllers
	
	Request URL: DELETE https://{CM-IP}:8003/controllers/
	
	Example schemas (JSON-based):
	
	INPUT:
	-
	
	OUTPUT:
	{
	"code": 200,
	"message": "Successfully deleted"
	}

GET controllers TSN	Returns a list with the information of all registered TSN-based controllers in the TSN Connectivity Manager
	
	Request URL: GET https://{CM-IP}:8003/controllers/tsn
	
	Example schemas (JSON-based):
	
	INPUT:
	-
	
	OUTPUT:
	
	[
	{
	"controller_id": "string",
	"name": "string",
	"description": "string",
	"url": "string",
	"port": 0,
	"username": "string",
	"password": "string",
	"type": "string"
	}
	]

GET controllers metro	Returns a list with the information of all registered metro controllers in the TSN Connectivity Manager Request URL: GET https://{CM-IP}:8003/controllers/metro Example schemas (JSON-based): INPUT:

	-
	
	OUTPUT:
	
	[
	  {
	  "controller_id": "string",
	  "name": "string",
	  "description": "string",
	  "url": "string",
	  "port": 0,
	  "username": "string",
	  "password": "string",
	  "type": "string"
	  }
	]

GET controllers {id}	Returns the information of a given controller registered in the TSN CM system with a concrete id.
	
	Request URL: GET https://{CM-IP}:8003/controllers/{id}
	
	Example schemas (JSON-based):
	
	INPUT:
	-
	
	OUTPUT:
	
	{
	  "controller_id": "string",
	  "name": "string",
	  "description": "string",
	  "url": "string",
	  "port": 0,
	  "username": "string",
	  "password": "string",
	  "type": "string"
	}

DELETE controller {id}	Delete from the TSN CM system the registered controller associated to a concrete id. Request URL: DELETE https://{CM-IP}:8003/controllers/{id} Example schemas (JSON-based): INPUT: - OUTPUT: { "code": 200,

<table>
<tr><td></td><td><pre>
  "message": "Successfully deleted"
}
</pre></td></tr>
<tr><td colspan="2">/flows</td></tr>
<tr><td>POST flows</td><td>Creates a new flow from a source to a destination node in a TSN network based on the body data. The query returns information about the new defined TSN flow

Request URL: POST https://{CM-IP}:8003/flows/

Example schemas (JSON-based):

INPUT:<pre>
{
  "name": "string",
  "description": "string",
  "topology_id": "string",
  "hostA": "string",
  "portA": "string",
  "hostB": "string",
  "portB": "string",
  "switchA": "string",
  "switchB": "string",
  "path": [
    "string"
  ],
  "trafficMix": {
    "video": {
      "scale": 0,
      "pattern": {
        "period": 0,
        "unit": "string",
        "pattern": [
          0
        ],
        "pattern_unit": "string"
      }
    },
    "gaming": {
      "scale": 0,
      "pattern": {
        "period": 0,
        "unit": "string",
        "pattern": [
          0
        ],
        "pattern_unit": "string"
      }
    },
    "internet": {
      "scale": 0,
      "pattern": {
        "period": 0,
        "unit": "string",
        "pattern": [
          0
        ],
        "pattern_unit": "string"
      }
</pre></td></tr>
</table>

```
    }
  },
  "bandwidth": 0,
  "latency": 0
}
```


OUTPUT:


```
{
  "flow_id": "string",
  "name": "string",
  "description": "string",
  "topology_id": "string",
  "hostA": "string",
  "portA": "string",
  "hostB": "string",
  "portB": "string",
  "switchA": "string",
  "switchB": "string",
  "path": [
    "string"
  ],
  "trafficMix": {
    "video": {
      "scale": 0,
      "pattern": {
        "period": 0,
        "unit": "string",
        "pattern": [
          0
        ],
        "pattern_unit": "string"
      }
    },
    "gaming": {
      "scale": 0,
      "pattern": {
        "period": 0,
        "unit": "string",
        "pattern": [
          0
        ],
        "pattern_unit": "string"
      }
    },
    "internet": {
      "scale": 0,
      "pattern": {
        "period": 0,
        "unit": "string",
        "pattern": [
          0
        ],
        "pattern_unit": "string"
      }
    }
  },
  "bandwidth": 0,
  "latency": 0
}
```

GET flows	Returns a list with all the enrolled TSN flows
	
	Request URL: GET https://{CM-IP}:8003/flows/
	
	Example schemas (JSON-based):
	
	INPUT:
	-
	
	OUTPUT:
	[
	{
	"flow_id": "string",
	"name": "string",
	"description": "string",
	"topology_id": "string",
	"hostA": "string",
	"portA": "string",
	"hostB": "string",
	"portB": "string",
	"switchA": "string",
	"switchB": "string",
	"path": [
	"string"
	],
	"trafficMix": {
	"video": {
	"scale": 0,
	"pattern": {
	"period": 0,
	"unit": "string",
	"pattern": [
	0
	],
	"pattern_unit": "string"
	}
	},
	"gaming": {
	"scale": 0,
	"pattern": {
	"period": 0,
	"unit": "string",
	"pattern": [
	0
	],
	"pattern_unit": "string"
	}
	},
	"internet": {
	"scale": 0,
	"pattern": {
	"period": 0,
	"unit": "string",
	"pattern": [
	0
	],
	"pattern_unit": "string"
	}
	}

<table>
<tr><td></td><td>
<pre>
    },
    "bandwidth": 0,
    "latency": 0
  }
]
</pre>
</td></tr>
<tr><td>DELETE controllers</td><td>Deletes all the registered TSN flows

Request URL: DELETE https://{CM-IP}:8003/flows/

Example schemas (JSON-based):

INPUT:
-

OUTPUT:
<pre>
{
  "code": 200,
  "message": "Successfully deleted"
}
</pre>
</td></tr>
<tr><td>GET flows {id}</td><td>Returns updated information of a TSN flow registered by a concrete id

Request URL: GET https://{CM-IP}:8003/flows/{id}

Example schemas (JSON-based):

INPUT:
-

OUTPUT:
<pre>
{
  "flow_id": "string",
  "name": "string",
  "description": "string",
  "topology_id": "string",
  "hostA": "string",
  "portA": "string",
  "hostB": "string",
  "portB": "string",
  "switchA": "string",
  "switchB": "string",
  "path": [
    "string"
  ],
  "trafficMix": {
    "video": {
      "scale": 0,
      "pattern": {
        "period": 0,
        "unit": "string",
        "pattern": [
          0
        ],
        "pattern_unit": "string"
      }
</pre>
</td></tr>
</table>

<table>
<tr><td></td><td>
<pre>
    },
    "gaming": {
      "scale": 0,
      "pattern": {
        "period": 0,
        "unit": "string",
        "pattern": [
          0
        ],
        "pattern_unit": "string"
      }
    },
    "internet": {
      "scale": 0,
      "pattern": {
        "period": 0,
        "unit": "string",
        "pattern": [
          0
        ],
        "pattern_unit": "string"
      }
    }
  },
  "bandwidth": 0,
  "latency": 0
}
</pre>
</td></tr>
<tr><td>DELETE flows {id}</td><td>Deletes an existing TSN flow registered by a concrete id

Request URL: DELETE https://{CM-IP}:8003/flows/{id}

Example schemas (JSON-based):

INPUT:
-

OUTPUT:
<pre>
{
  "code": 200,
  "message": "Successfully deleted"
}
</pre>
</td></tr>
<tr><td colspan="2">/topologies</td></tr>
<tr><td>GET topologies /all</td><td>Returns all the information about the underlaying network topologies in different segments and domains obtained by the previously registered controllers by executing a topology discovery operation.

Request URL: GET https://{CM-IP}:8003/topologies/all

Example schemas (JSON-based):

INPUT:
-</td></tr>
</table>

	OUTPUT:
	[
	  {
	  "topology_id": "string",
	  "source_controller": "string",
	  "network": [
	  {
	  "name": "string",
	  "description": "string",
	  "network_id": "string",
	  "network_types": {},
	  "node": [
	  {
	  "node_id": "string",
	  "name": "string",
	  "description": "string",
	  "type": "string",
	  "ports": {
	  "additionalProp1": {
	  "name": "string",
	  "port_type": "string",
	  "layer": "string",
	  "speed": 0,
	  "transmission_delay": 0
	  }
	  }
	  }
	  ],
	  "link": [
	  {
	  "link_id": "string",
	  "name": "string",
	  "description": "string",
	  "nodeA": "string",
	  "portA": "string",
	  "nodeB": "string",
	  "portB": "string",
	  "layer": "string",
	  "capacity": 0
	  }
	  ],
	  "l2_topology_attributes": {},
	  "l3_topology_attributes": {}
	  }
	  ]
	  }
	]

GET topologies tsn	Returns all the information about the underlaying network TSN topologies in different segments and domains obtained by the previously registered TSN controllers by executing a topology discovery operation. Request URL: GET https://{CM-IP}:8003/topologies/tsn Example schemas (JSON-based): INPUT: -

<table>
<tr><td></td><td>OUTPUT:
<pre>
[
  {
    "topology_id": "string",
    "source_controller": "string",
    "network": [
      {
        "name": "string",
        "description": "string",
        "network_id": "string",
        "network_types": {
            "layer": "string",
            "type": "TSN"
        },
        "node": [
          {
            "node_id": "string",
            "name": "string",
            "description": "string",
            "type": "string",
            "ports": {
              "P1": {
                "name": "string",
                "port_type": "string",
                "layer": "string",
                "speed": 0,
                "transmission_delay": 0
              }
            }
          }
        ],
        "link": [
          {
            "link_id": "string",
            "name": "string",
            "description": "string",
            "nodeA": "string",
            "portA": "string",
            "nodeB": "string",
            "portB": "string",
            "layer": "string",
            "capacity": 0
          }
        ],
        "l2_topology_attributes": {},
        "l3_topology_attributes": {}
      }
    ]
  }
]
</pre></td></tr>
<tr><td>GET topologies Metro</td><td>Returns all the information about the underlaying network TSN topologies in different segments and domains obtained by the previously registered TSN controllers by executing a topology discovery operation.

Request URL: GET https://{CM-IP}:8003/topologies/metro

Example schemas (JSON-based):</td></tr>
</table>

	INPUT:
	-
	
	OUTPUT:
	[
	  {
	  "topology_id": "string",
	  "source_controller": "string",
	  "network": [
	  {
	  "name": "string",
	  "description": "string",
	  "network_id": "string",
	  "network_types": {
	  "layer": "string",
	  "type": "metro"
	  },
	  "node": [
	  {
	  "node_id": "string",
	  "name": "string",
	  "description": "string",
	  "type": "string",
	  "ports": {
	  "P1": {
	  "name": "string",
	  "port_type": "string",
	  "layer": "string",
	  "speed": 0,
	  "transmission_delay": 0
	  }
	  }
	  }
	  ],
	  "link": [
	  {
	  "link_id": "string",
	  "name": "string",
	  "description": "string",
	  "nodeA": "string",
	  "portA": "string",
	  "nodeB": "string",
	  "portB": "string",
	  "layer": "string",
	  "capacity": 0
	  }
	  ],
	  "l2_topology_attributes": {},
	  "l3_topology_attributes": {}
	  }
	  ]
	  }
	]

GET topologies {id}	Returns all the information about the underlaying network TSN topologies in different segments and domains obtained by the previously registered TSN controllers by executing a topology discovery operation.

Request URL: GET https://{CM-IP}:8003/topologies/{ID}

Example schemas (JSON-based):

INPUT:
-

OUTPUT:


```
{
  "topology_id": "string",
  "source_controller": "string",
  "network": [
    {
      "name": "string",
      "description": "string",
      "network_id": "string",
      "network_types": {
          "layer": "string",
          "type": "string"
      },
      "node": [
        {
          "node_id": "string",
          "name": "string",
          "description": "string",
          "type": "string",
          "ports": {
            "P1": {
              "name": "string",
              "port_type": "string",
              "layer": "string",
              "speed": 0,
              "transmission_delay": 0
            }
          }
        }
      ],
      "link": [
        {
          "link_id": "string",
          "name": "string",
          "description": "string",
          "nodeA": "string",
          "portA": "string",
          "nodeB": "string",
          "portB": "string",
          "layer": "string",
          "capacity": 0
        }
      ],
      "l2_topology_attributes": {},
      "l3_topology_attributes": {}
    }
  ]
}
```

2.5.4 Stand-alone preliminary tests

To facilitate the user interaction with the Connectivity Manager NBI, we have developed a graphical user interface (see Figure 2-11**Error! Reference source not found.**) , publicly available, within the TIMING project scope, in the following URL: https://nbi-connectivity-manager.e-lighthouse.com/docs.

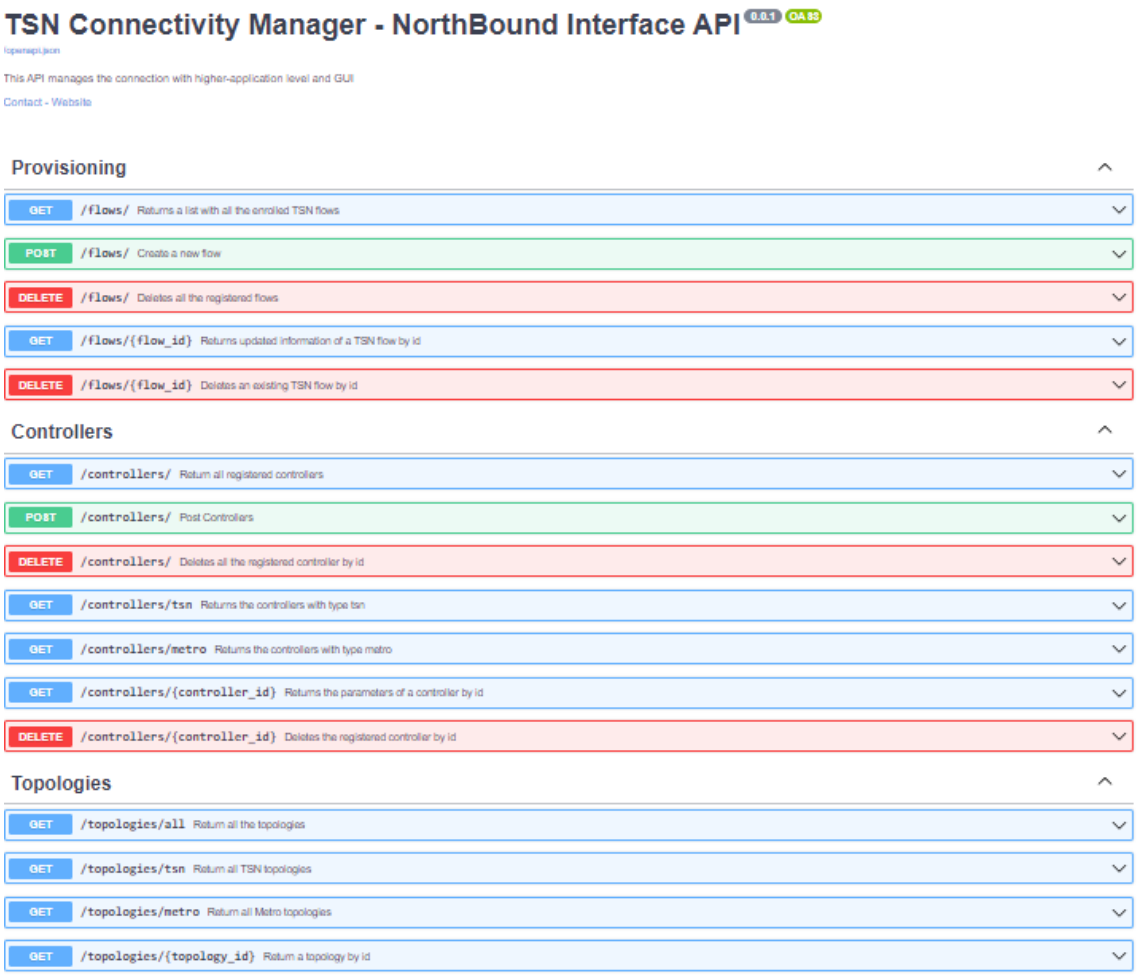


Figure 2-11: TSN Connectivity Manager NBI GUI

In addition, to validate the remote access to the functionalities provided by this API, a wide range of tests have been executed, classified according to their functional group. These tests have been performed using the POSTMAN software from a personal computer remotely located from the server on which the TSN CM is deployed.

- *Controllers*:

Functionality Test	**Test result**

POST controller	*Test 1:* Register a TSN controller:
	
	REQUEST
	`POST /controllers/ HTTP/1.1`
	`Content-Type: application/json`
	`User-Agent: PostmanRuntime/7.36.1`

```
Accept: */*
Postman-Token: 4fec6d7c-62c4-4b9b-971e-ec788614bfe9
Host: nbi-connectivity-manager.e-lighthouse.com
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 184

{
"name": "tsn_controller",
"description": "tsn_controller",
"url": "https://192.0.22.1",
"port": 8080,
"username": "user",
"password": "pass123",
"type": "tsn"
}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Length: 202
Content-Type: application/json
Date: Fri, 12 Jan 2024 07:39:54 GMT
Server: uvicorn

{"controller_id":"42f98df2-b54f-495f-8f31-
5e76b91f1fef","name":"tsn_controller","description":"tsn
_controller","url":"https://192.0.22.1","port":8080,"use
rname":"user","password":"pass123","type":"tsn"}
```

Test 2: Register a Metro controller:

REQUEST

```
POST /controllers/ HTTP/1.1
Content-Type: application/json
User-Agent: PostmanRuntime/7.36.1
Accept: */*
Postman-Token: 4b04e30e-659c-4820-b369-4b485f970699
Host: nbi-connectivity-manager.e-lighthouse.com
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Length: 188

{
"name": "metro_controller",
"description": "metro_controller",
"url": "https://192.0.46.1",
"port": 80,
"username": "user",
"password": "pass123",
"type": "metro"
}
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Length: 206
Content-Type: application/json
Date: Fri, 12 Jan 2024 07:40:25 GMT
Server: uvicorn
```

	{"controller_id":"a8aff4e3-6194-42c0-9b8f-b08ea8e39ecd","name":"metro_controller","description":"metro_controller","url":"https://192.0.46.1","port":80,"username":"user","password":"pass123","type":"metro"}
GET controllers	REQUEST GET /controllers/ HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: 0d71ef99-01f7-4b61-ad3a-4d0aba49673e Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 411 Content-Type: application/json Date: Fri, 12 Jan 2024 07:55:49 GMT Server: uvicorn [{"controller_id":"42f98df2-b54f-495f-8f31-5e76b91f1fef","name":"tsn_controller","description":"tsn_controller","url":"https://192.0.22.1","port":8080,"username":"user","password":"pass123","type":"tsn"},{"controller_id":"a8aff4e3-6194-42c0-9b8f-b08ea8e39ecd","name":"metro_controller","description":"metro_controller","url":"https://192.0.46.1","port":80,"username":"user","password":"pass123","type":"metro"}]
GET TSN controllers	REQUEST GET /controllers/tsn HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: d1706120-3903-4eb4-bff1-23f13d454785 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 204 Content-Type: application/json Date: Fri, 12 Jan 2024 09:33:10 GMT Server: uvicorn [{"controller_id":"42f98df2-b54f-495f-8f31-5e76b91f1fef","name":"tsn_controller","description":"tsn_controller","url":"https://192.0.22.1","port":8080,"username":"user","password":"pass123","type":"tsn"}]
GET Metro controllers	REQUEST GET /controllers/metro HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: 8db62c95-449c-44b9-b15a-4cfd7b3770d3 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive

	RESPONSE HTTP/1.1 200 OK Content-Length: 208 Content-Type: application/json Date: Fri, 12 Jan 2024 09:34:19 GMT Server: uvicorn [{"controller_id":"a8aff4e3-6194-42c0-9b8f-b08ea8e39ecd","name":"metro_controller","description":"metro_controller","url":"https://192.0.46.1","port":80,"username":"user","password":"pass123","type":"metro"}]
GET controller by ID	REQUEST GET /controllers/42f98df2-b54f-495f-8f31-5e76b91f1fef HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: ec2b90fb-69f3-497f-86c0-555afb8384d2 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 202 Content-Type: application/json Date: Fri, 12 Jan 2024 09:37:28 GMT Server: uvicorn {"controller_id":"42f98df2-b54f-495f-8f31-5e76b91f1fef","name":"tsn_controller","description":"tsn_controller","url":"https://192.0.22.1","port":8080,"username":"user","password":"pass123","type":"tsn"}

DELETE controller by ID	*Step 1: register a generic controller*
	
	REQUEST
	POST /controllers/ HTTP/1.1
	Content-Type: application/json
	User-Agent: PostmanRuntime/7.36.1
	Accept: */*
	Postman-Token: 2233e02d-7468-4ed6-9e94-e21bae77b4bc
	Host: nbi-connectivity-manager.e-lighthouse.com
	Accept-Encoding: gzip, deflate, br
	Connection: keep-alive
	Content-Length: 203
	
	{
	"name": "test-controller",
	"description": "test-controller",
	"url": "https://localhost",
	"port": 8080,
	"username": "user",
	"password": "pass123",
	"type": "default"
	}
	
	RESPONSE
	HTTP/1.1 200 OK
	Content-Length: 207
	Content-Type: application/json

	Date: Fri, 12 Jan 2024 09:29:32 GMT Server: uvicorn {"controller_id":"e62c9916-4007-499c-acae-cdd78ba1727f","name":"test-controller","description":"test-controller","url":"https://localhost","port":8080,"username":"user","password":"pass123","type":"default"} *Step 2: delete such controller by using the ID* REQUEST DELETE /controllers/e62c9916-4007-499c-acae-cdd78ba1727f HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: 0d683aaf-183f-43e0-9757-e989dc0d9de3 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 45 Content-Type: application/json Date: Fri, 12 Jan 2024 09:31:22 GMT Server: uvicorn {"code":200,"message":"Successfully deleted"}
DELETE all controllers	*Step 1: Delete all registered controllers* REQUEST DELETE /controllers/ HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: 396a9d22-73bf-4f05-8086-bee072505da8 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 45 Content-Type: application/json Date: Fri, 12 Jan 2024 09:38:38 GMT Server: uvicorn {"code":200,"message":"Successfully deleted"} Step 2: Check controllers registry REQUEST GET /controllers/ HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: c9d60739-a93b-4b56-aa3e-8e9f39306e7f Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br

	Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 2 Content-Type: application/json Date: Fri, 12 Jan 2024 09:40:26 GMT Server: uvicorn []

- *Topologies*:

Functionality Test	**Test result**
GET topologies/all	HTTP/1.1 200 OK Content-Length: 8243 Content-Type: application/json Date: Fri, 19 Jan 2024 09:35:57 GMT Server: uvicorn [{"topology_id":"42f98df2-b54f-495f-8f31-5e76b91f1fef","source_controller":"tsn-controller","network":[{"name":"TSN network","description":"TSN network as example","network_id":"8849560a-462a-4992-97e2-12ad8d781ded","network_types":{"layer":"L2","type":"TSN"},"node":[{"node_id":"2113f1da-345f-4276-a99b-2a5ab1b3626b","name":"H1","description":"Host 1","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L2","speed":100,"transmission_delay":null}}},{"node_id":"7f39f1fa-e971-47db-81cb-60a79524b08f","name":"H2","description":"Host 2","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L2","speed":100,"transmission_delay":null}}},{"node_id":"1acb8de8-9983-4222-bbef-0c65c96afc65","name":"H3","description":"Host 3","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L2","speed":100,"transmission_delay":null}}},{"node_id":"fa7c3854-77c1-46a3-b6e7-6b3a157e2272","name":"H4","description":"Host 4","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L2","speed":100,"transmission_delay":null}}},{"node_id":"e15946e1-6128-49db-ad70-9954e41035a1","name":"R1","description":"Switch 1","type":"switch","ports":{"A1":{"name":"A1","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"A2":{"name":"A2","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"A3":{"name":"A3","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"T1":{"name":"T1","port_type":"trunk","layer":"L2","speed":100,"transmission_delay":60},"T2":{"name":"T2","port_type":"trunk","layer":"L2","speed":1,"transmission_delay":60}}},{"node_id":"96a12a72-957e-4c06-8c61-7ecb1e2fae76","name":"R2","description":"Switch 2","type":"switch","ports":{"A1":{"name":"A1","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"A2":{"name":"A2","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"A3":{"name":"A3","port_type":"access","layer":"L2","speed":100,"trans

mission_delay":null},"T1":{"name":"T1","port_type":"trunk","layer":"L2","speed":100,"transmission_delay":60},"T2":{"name":"T2","port_type":"trunk","layer":"L2","speed":1,"transmission_delay":60}}},{"node_id":"24b825b0-5826-4628-982f-b008379ed956","name":"R3","description":"Switch 3","type":"switch","ports":{"A1":{"name":"A1","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"A2":{"name":"A2","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"A3":{"name":"A3","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"T1":{"name":"T1","port_type":"trunk","layer":"L2","speed":100,"transmission_delay":60},"T2":{"name":"T2","port_type":"trunk","layer":"L2","speed":1,"transmission_delay":60}}}],"link":[{"link_id":"d2b75c6f-eeed-4328-bd30-f58cc87bf041","name":"L_H1R1","description":"Link between H1 and R1","nodeA":"H1","portA":"E1","nodeB":"R1","portB":"A1","layer":"L2","capacity":100},{"link_id":"85810507-24dc-485c-afcf-0821de0a6102","name":"L_H2R2","description":"Link between H2 and R2","nodeA":"H2","portA":"E1","nodeB":"R2","portB":"A1","layer":"L2","capacity":100},{"link_id":"44a5ba71-7ed1-467c-b406-364186c7ec34","name":"L_R3H3","description":"Link between R3 and H3","nodeA":"R3","portA":"A1","nodeB":"H3","portB":"E1","layer":"L2","capacity":100},{"link_id":"2eab7157-788d-49bc-965a-fde559a8b952","name":"L_R3H4","description":"Link between R3 and H4","nodeA":"R3","portA":"A2","nodeB":"H4","portB":"E1","layer":"L2","capacity":100},{"link_id":"f2c0f714-274b-4d10-a46b-c465e9bc1b3d","name":"L_R1R2","description":"Link between R1 and R2","nodeA":"R1","portA":"T1","nodeB":"R2","portB":"T1","layer":"L2","capacity":100},{"link_id":"d7c8575f-0ba3-4a54-a785-b87c2aeea6c3","name":"L_R1R3","description":"Link between R1 and R3","nodeA":"R1","portA":"T2","nodeB":"R3","portB":"T1","layer":"L2","capacity":100},{"link_id":"2ffcc5b0-ea7f-48af-9c7b-576e70c7322f","name":"L_R2R3","description":"Link between R2 and R3","nodeA":"R2","portA":"T2","nodeB":"R3","portB":"T2","layer":"L2","capacity":100}],"l2_topology_attributes":{},"l3_topology_attributes":{}}]},{"topology_id":"a8aff4e3-6194-42c0-9b8f-b08ea8e39ecd","source_controller":"metro-controller","network":[{"name":"Metro network","description":"Metro network as example","network_id":"e856f877-2ea7-45a7-bdb0-a8cdb10ee745","network_types":{"layer":"L2","type":"metro"},"node":[{"node_id":"6ef46f62-1ee6-40e8-b67f-2b9ae005099a","name":"H1","description":"Host 1","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L2","speed":100,"transmission_delay":null}}},{"node_id":"36d16961-24ef-4c23-9303-dfe4e34c1a93","name":"H2","description":"Host 2","type":"host","ports":{"E1":{"name":"E1","port_type":

"host","layer":"L2","speed":100,"transmission_delay":nul
l}}},{"node_id":"188338c3-c769-482b-b181-
4b6d2414be95","name":"H3","description":"Host
3","type":"host","ports":{"E1":{"name":"E1","port_type":
"host","layer":"L2","speed":100,"transmission_delay":nul
l}}},{"node_id":"c8155352-5c7c-4334-998a-
56b8169fd191","name":"H4","description":"Host
4","type":"host","ports":{"E1":{"name":"E1","port_type":
"host","layer":"L2","speed":100,"transmission_delay":nul
l}}},{"node_id":"e5ed148c-c32e-4cb5-8bf5-
89f25f011d22","name":"R1","description":"Switch
1","type":"switch","ports":{"A1":{"name":"A1","port_type
":"access","layer":"L2","speed":100,"transmission_delay"
:null},"A2":{"name":"A2","port_type":"access","layer":"L
2","speed":100,"transmission_delay":null},"A3":{"name":"
A3","port_type":"access","layer":"L2","speed":100,"trans
mission_delay":null},"T1":{"name":"T1","port_type":"trun
k","layer":"L2","speed":100,"transmission_delay":60},"T2
":{"name":"T2","port_type":"trunk","layer":"L2","speed":
1,"transmission_delay":60}}},{"node_id":"8bbc50a0-598a-
458e-84bd-
193c5a74b250","name":"R2","description":"Switch
2","type":"switch","ports":{"A1":{"name":"A1","port_type
":"access","layer":"L2","speed":100,"transmission_delay"
:null},"A2":{"name":"A2","port_type":"access","layer":"L
2","speed":100,"transmission_delay":null},"A3":{"name":"
A3","port_type":"access","layer":"L2","speed":100,"trans
mission_delay":null},"T1":{"name":"T1","port_type":"trun
k","layer":"L2","speed":100,"transmission_delay":60},"T2
":{"name":"T2","port_type":"trunk","layer":"L2","speed":
1,"transmission_delay":60}}},{"node_id":"8a5e07a7-9e0e-
489f-b7f7-
6fb5ec1722f7","name":"R3","description":"Switch
3","type":"switch","ports":{"A1":{"name":"A1","port_type
":"access","layer":"L2","speed":100,"transmission_delay"
:null},"A2":{"name":"A2","port_type":"access","layer":"L
2","speed":100,"transmission_delay":null},"A3":{"name":"
A3","port_type":"access","layer":"L2","speed":100,"trans
mission_delay":null},"T1":{"name":"T1","port_type":"trun
k","layer":"L2","speed":100,"transmission_delay":60},"T2
":{"name":"T2","port_type":"trunk","layer":"L2","speed":
1,"transmission_delay":60}}}],"link":[{"link_id":"8da29f
75-effe-427a-8f50-
e5fdbbdddb03","name":"L_H1R1","description":"Link
between H1 and
R1","nodeA":"H1","portA":"E1","nodeB":"R1","portB":"A1",
"layer":"L2","capacity":100},{"link_id":"d33bd436-978f-
4733-bc15-
378d2795278a","name":"L_H2R2","description":"Link
between H2 and
R2","nodeA":"H2","portA":"E1","nodeB":"R2","portB":"A1",
"layer":"L2","capacity":100},{"link_id":"64eb06f2-2fc7-
4194-80cd-
313f96a17886","name":"L_R3H3","description":"Link
between R3 and
H3","nodeA":"R3","portA":"A1","nodeB":"H3","portB":"E1",
"layer":"L2","capacity":100},{"link_id":"4e86ca02-06d0-
4ca6-9091-
8d892d00e7ef","name":"L_R3H4","description":"Link
between R3 and
H4","nodeA":"R3","portA":"A2","nodeB":"H4","portB":"E1",
"layer":"L2","capacity":100},{"link_id":"5564edc8-a881-
4751-bd64-
bfae68ea3f48","name":"L_R1R2","description":"Link

	between R1 and R2","nodeA":"R1","portA":"T1","nodeB":"R2","portB":"T1", "layer":"L2","capacity":100},{"link_id":"ddd837fa-973a-4ec8-a1c6-57d26a650469","name":"L_R1R3","description":"Link between R1 and R3","nodeA":"R1","portA":"T2","nodeB":"R3","portB":"T1", "layer":"L2","capacity":100},{"link_id":"ece732f6-a3ff-47fb-a255-929e7dfe3c0b","name":"L_R2R3","description":"Link between R2 and R3","nodeA":"R2","portA":"T2","nodeB":"R3","portB":"T2", "layer":"L2","capacity":100}],"l2_topology_attributes":{},"l3_topology_attributes":{}}]}]
GET topologies/tsn	REQUEST GET /topologies/tsn HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: e4ec08df-9ef8-423f-857a-420d1fa21363 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 5165 Content-Type: application/json Date: Fri, 19 Jan 2024 07:57:20 GMT Server: uvicorn [{"topology_id":"42f98df2-b54f-495f-8f31-5e76b91f1fef","source_controller":"tsn-controller","network":[{"name":"TSN network","description":"TSN network of employees at ELIG","network_id":"9beee57a-b67e-4c20-9c9e-2914f3bdad63","network_types":{"layer":"L2","type":"TSN"},"node":[{"node_id":"2891b00f-6ec2-4f20-831b-ccf4b72895f7","name":"switch1","description":"Switch 1 located at Vigo","type":"switch","ports":{"A1":{"name":"A1","port_type":"access","layer":"L2","speed":20,"transmission_delay":null},"A2":{"name":"A2","port_type":"access","layer":"L2","speed":20,"transmission_delay":2},"A3":{"name":"A3","port_type":"access","layer":"L2","speed":20,"transmission_delay":null},"A4":{"name":"A4","port_type":"access","layer":"L2","speed":20,"transmission_delay":2},"T1":{"name":"T1","port_type":"trunk","layer":"L2","speed":100,"transmission_delay":7},"T2":{"name":"T2","port_type":"trunk","layer":"L2","speed":1,"transmission_delay":7}}},{"node_id":"76b87550-acb4-4efe-a345-5beb6300b817","name":"switch2","description":"Switch 2 located at Cartagena","type":"switch","ports":{"A1":{"name":"A1","port_type":"access","layer":"L2","speed":20,"transmission_delay":null},"A2":{"name":"A2","port_type":"access","layer":"L2","speed":20,"transmission_delay":2},"A3":{"name":"A3","port_type":"access","layer":"L2","speed":20,"transmission_delay":null},"A4":{"name":"A4","port_type":"access","layer":"L2","speed":20,"transmission_delay":2},"T1":{"name":"T1","port_type":"trunk","layer":"L2","speed":100,"transmission_delay":7},"T2":{"name":"T2","port_type":"trunk","layer":"L2","speed":1,"transmission_delay":7

}}},{"node_id":"479d4dce-7954-473e-963c-0939fa193921","name":"switch3","description":"Switch 3 located at Malaga","type":"switch","ports":{"A1":{"name":"A1","port_type":"access","layer":"L2","speed":20,"transmission_delay":null},"A2":{"name":"A2","port_type":"access","layer":"L2","speed":20,"transmission_delay":2},"A3":{"name":"A3","port_type":"access","layer":"L2","speed":20,"transmission_delay":null},"A4":{"name":"A4","port_type":"access","layer":"L2","speed":20,"transmission_delay":2},"T1":{"name":"T1","port_type":"trunk","layer":"L2","speed":100,"transmission_delay":7},"T2":{"name":"T2","port_type":"trunk","layer":"L2","speed":1,"transmission_delay":7}}},{"node_id":"a38dd09c-028a-432f-a04d-330b461920ed","name":"switch4","description":"Switch 4 located at Alicante","type":"switch","ports":{"A1":{"name":"A1","port_type":"access","layer":"L2","speed":20,"transmission_delay":null},"A2":{"name":"A2","port_type":"access","layer":"L2","speed":20,"transmission_delay":2},"A3":{"name":"A3","port_type":"access","layer":"L2","speed":20,"transmission_delay":null},"A4":{"name":"A4","port_type":"access","layer":"L2","speed":20,"transmission_delay":2},"T1":{"name":"T1","port_type":"trunk","layer":"L2","speed":100,"transmission_delay":7},"T2":{"name":"T2","port_type":"trunk","layer":"L2","speed":1,"transmission_delay":7}}}],"link":[{"link_id":"7253e9c9-8b8e-457e-989c-8c65a6c88859","name":"link_vc","description":"Link between Vigo and Cartagena","nodeA":"switch1","portA":"T1","nodeB":"switch2","portB":"T1","layer":"L2","capacity":100},{"link_id":"8d73ad67-fe5c-455a-9458-9b4d1940ad87","name":"link_cm","description":"Link between Cartagena and Malaga","nodeA":"switch2","portA":"T2","nodeB":"switch3","portB":"T2","layer":"L2","capacity":100},{"link_id":"8dec6cce-ce01-4d8a-9c70-a488345e4acd","name":"link_ma","description":"Link between Malaga and Alicante","nodeA":"switch3","portA":"T1","nodeB":"switch4","portB":"T1","layer":"L2","capacity":100},{"link_id":"5e5b8682-215c-489d-94c1-7e3a758f2178","name":"link_av","description":"Link between Alicante and Vigo","nodeA":"switch4","portA":"T2","nodeB":"switch1","portB":"T2","layer":"L2","capacity":100}],"l2_topology_attributes":{},"l3_topology_attributes":{}},{"name":"TSN network","description":"TSN network of employees at ELIG","network_id":"3e5243f3-675f-4a72-a69b-f3d877f76004","network_types":{"layer":"L3","type":"TSN"},"node":[{"node_id":"e693930f-f5b6-4fe6-8c86-270097ff35a7","name":"host1","description":"Host 1 located at Vigo","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L3","speed":100,"transmission_delay":null}}},{"node_id":"28ed0288-6356-40a4-ac86-2e6bdc02951a","name":"host2","description":"Host 2 located at Cartagena","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L3","speed":100,"transmission_delay":null}}},{"node_id":"45ae8e51-fcd2-49d4-ae98-71c4bfd71b2e","name":"host3","description":"Host 3 located at Malaga","type":"host","ports":{"E1":{"name":"E1","port_t

	ype":"host","layer":"L3","speed":100,"transmission_delay":null}}},{"node_id":"77425b92-f7aa-4f4e-90d1-6bbb33ff5b03","name":"host4","description":"Host 4 located at Alicante","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L3","speed":100,"transmission_delay":null}}}],"link":[{"link_id":"f2dbf1d7-39a9-449c-8257-514616570727","name":"link_vc_v","description":"Virtual link between Vigo and Cartagena","nodeA":"host1","portA":"E1","nodeB":"host2","portB":"E1","layer":"L3","capacity":100}],"l2_topology_attributes":{},"l3_topology_attributes":{}}]}]
GET topologies/metro	REQUEST GET /topologies/tsn HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: d91e9f46-4e4d-4c73-a29a-91dfd8b7e2c6 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 4118 Content-Type: application/json Date: Fri, 19 Jan 2024 09:37:05 GMT Server: uvicorn [{"topology_id":"42f98df2-b54f-495f-8f31-5e76b91f1fef","source_controller":"tsn-controller","network":[{"name":"TSN network","description":"TSN network as example","network_id":"5e230bed-e576-4262-8009-c31edcfc3bd3","network_types":{"layer":"L2","type":"TSN"},"node":[{"node_id":"c4c7a626-a344-4aef-ac64-54e1c3cc5ca3","name":"H1","description":"Host 1","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L2","speed":100,"transmission_delay":null}}},{"node_id":"3c58f9e8-6d76-458e-9625-03b9218ce9e3","name":"H2","description":"Host 2","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L2","speed":100,"transmission_delay":null}}},{"node_id":"02787165-3438-48ad-bf00-c655826762a9","name":"H3","description":"Host 3","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L2","speed":100,"transmission_delay":null}}},{"node_id":"2f587ce4-32cc-43a6-87da-499490b66bdc","name":"H4","description":"Host 4","type":"host","ports":{"E1":{"name":"E1","port_type":"host","layer":"L2","speed":100,"transmission_delay":null}}},{"node_id":"054c0cd0-8cda-44a1-b530-7b8a9cdaecdc","name":"R1","description":"Switch 1","type":"switch","ports":{"A1":{"name":"A1","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"A2":{"name":"A2","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"A3":{"name":"A3","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"T1":{"name":"T1","port_type":"trunk","layer":"L2","speed":100,"transmission_delay":60},"T2":{"name":"T2","port_type":"trunk","layer":"L2","speed":1,"transmission_delay":60}}},{"node_id":"23905228-f5dc-

	4642-be3c- 9dd0adc62034","name":"R2","description":"Switch 2","type":"switch","ports":{"A1":{"name":"A1","port_type ":"access","layer":"L2","speed":100,"transmission_delay" :null},"A2":{"name":"A2","port_type":"access","layer":"L 2","speed":100,"transmission_delay":null},"A3":{"name":" A3","port_type":"access","layer":"L2","speed":100,"trans mission_delay":null},"T1":{"name":"T1","port_type":"trun k","layer":"L2","speed":100,"transmission_delay":60},"T2 ":{"name":"T2","port_type":"trunk","layer":"L2","speed": 1,"transmission_delay":60}}},{"node_id":"8e1ae5b3-82ab- 45aa-97f4- bd3ad35efc6e","name":"R3","description":"Switch 3","type":"switch","ports":{"A1":{"name":"A1","port_type ":"access","layer":"L2","speed":100,"transmission_delay" :null},"A2":{"name":"A2","port_type":"access","layer":"L 2","speed":100,"transmission_delay":null},"A3":{"name":" A3","port_type":"access","layer":"L2","speed":100,"trans mission_delay":null},"T1":{"name":"T1","port_type":"trun k","layer":"L2","speed":100,"transmission_delay":60},"T2 ":{"name":"T2","port_type":"trunk","layer":"L2","speed": 1,"transmission_delay":60}}}],"link":[{"link_id":"318d19 17-e97a-4555-ad90- b11bf2758715","name":"L_H1R1","description":"Link between H1 and R1","nodeA":"H1","portA":"E1","nodeB":"R1","portB":"A1", "layer":"L2","capacity":100},{"link_id":"67e54702-69da- 495a-a1ed- 75640692de58","name":"L_H2R2","description":"Link between H2 and R2","nodeA":"H2","portA":"E1","nodeB":"R2","portB":"A1", "layer":"L2","capacity":100},{"link_id":"2abd9dfb-8427- 48d9-beb7- 30911ac66d76","name":"L_R3H3","description":"Link between R3 and H3","nodeA":"R3","portA":"A1","nodeB":"H3","portB":"E1", "layer":"L2","capacity":100},{"link_id":"3fee4a79-146c- 45ac-b86c- 63fb8da5f3c8","name":"L_R3H4","description":"Link between R3 and H4","nodeA":"R3","portA":"A2","nodeB":"H4","portB":"E1", "layer":"L2","capacity":100},{"link_id":"1c04b12d-f360- 45b8-94cc- eccfd36cfa34","name":"L_R1R2","description":"Link between R1 and R2","nodeA":"R1","portA":"T1","nodeB":"R2","portB":"T1", "layer":"L2","capacity":100},{"link_id":"773cd022-223b- 4058-9e69- 4a1c172fa871","name":"L_R1R3","description":"Link between R1 and R3","nodeA":"R1","portA":"T2","nodeB":"R3","portB":"T1", "layer":"L2","capacity":100},{"link_id":"ec4df91b-f0c8- 4f3e-bcf5- 85cd8126772b","name":"L_R2R3","description":"Link between R2 and R3","nodeA":"R2","portA":"T2","nodeB":"R3","portB":"T2", "layer":"L2","capacity":100}],"l2_topology_attributes":{ },"l3_topology_attributes":{}}]}]
GET topologies by ID	REQUEST GET /topologies/a8aff4e3-6194-42c0-9b8f-b08ea8e39ecd HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */*

```
Postman-Token: 46f86606-7f4c-4373-8ca9-4526122bf085
Host: nbi-connectivity-manager.e-lighthouse.com
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```


RESPONSE


```
HTTP/1.1 200 OK
Content-Length: 5177
Content-Type: application/json
Date: Fri, 19 Jan 2024 07:58:36 GMT
Server: uvicorn

HTTP/1.1 200 OK
Content-Length: 4124
Content-Type: application/json
Date: Fri, 19 Jan 2024 09:38:48 GMT
Server: uvicorn

{"topology_id":"a8aff4e3-6194-42c0-9b8f-
b08ea8e39ecd","source_controller":"metro-
controller","network":[{"name":"Metro
network","description":"Metro network as
example","network_id":"13ebde75-7db8-4001-8395-
0ad40fbcd8d8","network_types":{"layer":"L2","type":"metr
o"},"node":[{"node_id":"f458555f-4a0c-43af-bb1f-
3b60e22620c4","name":"H1","description":"Host
1","type":"host","ports":{"E1":{"name":"E1","port_type":
"host","layer":"L2","speed":100,"transmission_delay":nul
l}}},{"node_id":"2b3ad396-100e-43af-8a20-
4c8c43c71781","name":"H2","description":"Host
2","type":"host","ports":{"E1":{"name":"E1","port_type":
"host","layer":"L2","speed":100,"transmission_delay":nul
l}}},{"node_id":"1eb3eff8-ea2a-4ae4-a8a5-
6dcd1d2ee52e","name":"H3","description":"Host
3","type":"host","ports":{"E1":{"name":"E1","port_type":
"host","layer":"L2","speed":100,"transmission_delay":nul
l}}},{"node_id":"a59cd41e-3b13-4ffa-87bc-
ead5da9b2174","name":"H4","description":"Host
4","type":"host","ports":{"E1":{"name":"E1","port_type":
"host","layer":"L2","speed":100,"transmission_delay":nul
l}}},{"node_id":"577c82c1-7117-4b86-8f22-
2b75a7e5f7a4","name":"R1","description":"Switch
1","type":"switch","ports":{"A1":{"name":"A1","port_type
":"access","layer":"L2","speed":100,"transmission_delay"
:null},"A2":{"name":"A2","port_type":"access","layer":"L
2","speed":100,"transmission_delay":null},"A3":{"name":"
A3","port_type":"access","layer":"L2","speed":100,"trans
mission_delay":null},"T1":{"name":"T1","port_type":"trun
k","layer":"L2","speed":100,"transmission_delay":60},"T2
":{"name":"T2","port_type":"trunk","layer":"L2","speed":
1,"transmission_delay":60}}},{"node_id":"9cfcc4ea-14bc-
4171-bbf0-
9cef91248d38","name":"R2","description":"Switch
2","type":"switch","ports":{"A1":{"name":"A1","port_type
":"access","layer":"L2","speed":100,"transmission_delay"
:null},"A2":{"name":"A2","port_type":"access","layer":"L
2","speed":100,"transmission_delay":null},"A3":{"name":"
A3","port_type":"access","layer":"L2","speed":100,"trans
mission_delay":null},"T1":{"name":"T1","port_type":"trun
k","layer":"L2","speed":100,"transmission_delay":60},"T2
":{"name":"T2","port_type":"trunk","layer":"L2","speed":
1,"transmission_delay":60}}},{"node_id":"cae3f4f8-c705-
41ba-959b-
bf1657cd645b","name":"R3","description":"Switch
```

	3","type":"switch","ports":{"A1":{"name":"A1","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"A2":{"name":"A2","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"A3":{"name":"A3","port_type":"access","layer":"L2","speed":100,"transmission_delay":null},"T1":{"name":"T1","port_type":"trunk","layer":"L2","speed":100,"transmission_delay":60},"T2":{"name":"T2","port_type":"trunk","layer":"L2","speed":1,"transmission_delay":60}}}],"link":[{"link_id":"d72c638e-d08d-4370-9a7f-f9a1c786e8cb","name":"L_H1R1","description":"Link between H1 and R1","nodeA":"H1","portA":"E1","nodeB":"R1","portB":"A1","layer":"L2","capacity":100},{"link_id":"4287077c-5e19-43d5-965a-01d46748e09f","name":"L_H2R2","description":"Link between H2 and R2","nodeA":"H2","portA":"E1","nodeB":"R2","portB":"A1","layer":"L2","capacity":100},{"link_id":"05ded8a8-260d-42dd-bdcd-fa2f74839a38","name":"L_R3H3","description":"Link between R3 and H3","nodeA":"R3","portA":"A1","nodeB":"H3","portB":"E1","layer":"L2","capacity":100},{"link_id":"3ef9118c-c03d-488a-8b67-1508a2efa233","name":"L_R3H4","description":"Link between R3 and H4","nodeA":"R3","portA":"A2","nodeB":"H4","portB":"E1","layer":"L2","capacity":100},{"link_id":"7959181d-6a5b-4862-8a7c-bcd2dc2fb688","name":"L_R1R2","description":"Link between R1 and R2","nodeA":"R1","portA":"T1","nodeB":"R2","portB":"T1","layer":"L2","capacity":100},{"link_id":"593dda1b-afb9-4b37-897d-2388276da826","name":"L_R1R3","description":"Link between R1 and R3","nodeA":"R1","portA":"T2","nodeB":"R3","portB":"T1","layer":"L2","capacity":100},{"link_id":"7a9e2048-d357-49de-9fdf-fd651d7ce7ac","name":"L_R2R3","description":"Link between R2 and R3","nodeA":"R2","portA":"T2","nodeB":"R3","portB":"T2","layer":"L2","capacity":100}],"l2_topology_attributes":{},"l3_topology_attributes":{}}]}

- Flows:

Functionality Test	Test result
POST flow	REQUEST POST /flows/ HTTP/1.1 Content-Type: application/json User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: a5a3bbf9-b130-4adf-971d-668cee88a398 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive Content-Length: 1004 {

```
"name": "'BG_Traffic1",
"description": "Testing Flow 1",
"topology_id": "42f98df2-b54f-495f-8f31-5e76b91f1fef",
"hostA": "H1",
"portA": "E1",
"hostB": "H3",
"portB": "E1",
"switchA": "R1",
"switchB": "R3",
"path": [
"[R1,R3]]"
],
"trafficMix": {
"video": {
"scale": 100,
"pattern": {
"period": 86400,
"unit": "s",
"pattern": [
0.25,0.3,0.32,0.35,0.39
],
"pattern_unit": "min"
}
},
"gaming": {
"scale": 300,
"pattern": {
"period": 86400,
"unit": "s",
"pattern": [
0.25,0.3,0.32,0.35,0.39
],
"pattern_unit": "min"
}
},
"internet": {
"scale": 500,
"pattern": {
"period": 86400,
"unit": "s",
"pattern": [
0.25,0.3,0.32,0.35,0.39
],
"pattern_unit": "min"
}
}
},
"bandwidth": 50,
"latency": 20
}
```


RESPONSE


```
HTTP/1.1 200 OK
Content-Length: 660
Content-Type: application/json
Date: Fri, 19 Jan 2024 09:46:52 GMT
Server: uvicorn

{"flow_id":"b5fb6aa0-0af2-41b7-ba71-
bb5cf522630f","name":"'BG_Traffic1","description":"Testi
ng Flow 1","topology_id":"42f98df2-b54f-495f-8f31-
5e76b91f1fef","hostA":"H1","portA":"E1","hostB":"H3","po
rtB":"E1","switchA":"R1","switchB":"R3","path":["[R1,R3]
]"],"trafficMix":{"video":{"scale":100,"pattern":{"perio
d":86400,"unit":"s","pattern":[0.25,0.3,0.32,0.35,0.39],
```

	"pattern_unit":"min"}},"gaming":{"scale":300,"pattern":{ "period":86400,"unit":"s","pattern":[0.25,0.3,0.32,0.35, 0.39],"pattern_unit":"min"}},"internet":{"scale":500,"pa ttern":{"period":86400,"unit":"s","pattern":[0.25,0.3,0. 32,0.35,0.39],"pattern_unit":"min"}}},"bandwidth":50.0," latency":20.0}
GET flows	REQUEST GET /flows/ HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: 3df0f890-68a4-4332-8b67-57bd87a12f82 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 1324 Content-Type: application/json Date: Fri, 19 Jan 2024 09:50:21 GMT Server: uvicorn [{"flow_id":"b5fb6aa0-0af2-41b7-ba71- bb5cf522630f","name":"'BG_Traffic1","description":"Testi ng Flow 1","topology_id":"42f98df2-b54f-495f-8f31- 5e76b91f1fef","hostA":"H1","portA":"E1","hostB":"H3","po rtB":"E1","switchA":"R1","switchB":"R3","path":["[R1,R3]]"],"trafficMix":{"video":{"scale":100,"pattern":{"perio d":86400,"unit":"s","pattern":[0.25,0.3,0.32,0.35,0.39], "pattern_unit":"min"}},"gaming":{"scale":300,"pattern":{ "period":86400,"unit":"s","pattern":[0.25,0.3,0.32,0.35, 0.39],"pattern_unit":"min"}},"internet":{"scale":500,"pa ttern":{"period":86400,"unit":"s","pattern":[0.25,0.3,0. 32,0.35,0.39],"pattern_unit":"min"}}},"bandwidth":50.0," latency":20.0},{"flow_id":"a9074d69-2f11-4574-8021- 01c4d8039a76","name":"'BG_Traffic2'","description":"Test ing Flow 2","topology_id":"42f98df2-b54f-495f-8f31- 5e76b91f1fef","hostA":"H2","portA":"E1","hostB":"H4","po rtB":"E1","switchA":"R2","switchB":"R3","path":["[R2,R3]]"],"trafficMix":{"video":{"scale":100,"pattern":{"perio d":86400,"unit":"s","pattern":[0.25,0.3,0.32,0.35,0.39], "pattern_unit":"min"}},"gaming":{"scale":300,"pattern":{ "period":86400,"unit":"s","pattern":[0.25,0.3,0.32,0.35, 0.39],"pattern_unit":"min"}},"internet":{"scale":500,"pa ttern":{"period":86400,"unit":"s","pattern":[0.25,0.3,0. 32,0.35,0.39],"pattern_unit":"min"}}},"bandwidth":50.0," latency":20.0}]
GET flows by ID	REQUEST GET /flows/b5fb6aa0-0af2-41b7-ba71-bb5cf522630f HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: 754dd7e2-0a0b-4b8e-abe7-4feac12782d0 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 662 Content-Type: application/json

	Date: Fri, 19 Jan 2024 09:50:59 GMT Server: uvicorn [{"flow_id":"b5fb6aa0-0af2-41b7-ba71-bb5cf522630f","name":"'BG_Traffic1","description":"Testing Flow 1","topology_id":"42f98df2-b54f-495f-8f31-5e76b91f1fef","hostA":"H1","portA":"E1","hostB":"H3","portB":"E1","switchA":"R1","switchB":"R3","path":["[R1,R3]"],"trafficMix":{"video":{"scale":100,"pattern":{"period":86400,"unit":"s","pattern":[0.25,0.3,0.32,0.35,0.39],"pattern_unit":"min"}},"gaming":{"scale":300,"pattern":{"period":86400,"unit":"s","pattern":[0.25,0.3,0.32,0.35,0.39],"pattern_unit":"min"}},"internet":{"scale":500,"pattern":{"period":86400,"unit":"s","pattern":[0.25,0.3,0.32,0.35,0.39],"pattern_unit":"min"}}},"bandwidth":50.0,"latency":20.0}]
DELETE flows by ID	REQUEST DELETE /flows/a9074d69-2f11-4574-8021-01c4d8039a76 HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: 9c416e9b-5075-4953-85ef-c7c83112579e Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 45 Content-Type: application/json Date: Fri, 19 Jan 2024 09:51:53 GMT Server: uvicorn {"code":200,"message":"Successfully deleted"}
DELETE flows	REQUEST DELETE /flows/ HTTP/1.1 User-Agent: PostmanRuntime/7.36.1 Accept: */* Postman-Token: a19a948c-c507-4110-8dbc-4a42c6144261 Host: nbi-connectivity-manager.e-lighthouse.com Accept-Encoding: gzip, deflate, br Connection: keep-alive RESPONSE HTTP/1.1 200 OK Content-Length: 45 Content-Type: application/json Date: Fri, 19 Jan 2024 09:52:26 GMT Server: uvicorn {"code":200,"message":"Successfully deleted"}

2.6 TSN MODELS/DT

2.6.1 Description

The TSN Digital Twin includes the following basic modules (see Figure 2-12):

- a manager module configuring and supervising the operation of the rest of the modules.
- a few modules that include algorithm, models, and the interface with the TSN connectivity manager.
- a Redis DB that is used in publish-subscribe mode to communicate the different modules among them.

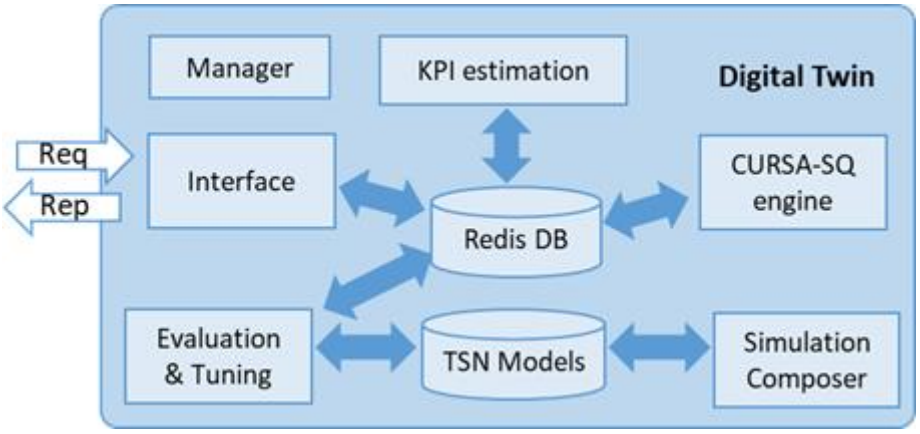


Figure 2-12 Internal architecture of the Digital Twin

In this deliverable, we report the work that has been done from the status reported in TIMING SP-1 D1.3, especially regarding integration. In particular:

- Definition of common traffic profiles based on a set of defined service profiles and traffic patterns that will be adopted by other systems in the TIMING architecture, specifically in the TSN Connectivity Manager.
- Advances on the development of a preliminary version of the REST-API that now is integrated with the other internal modules of the DT.
- Development of a *Simulation Composer* module that creates the simulation scenario based on the topology and KPI evaluation request received.

2.6.2 Traffic Profiles

Two descriptors have been identified to define traffic profiles: service profiles and traffic patterns.

As for service profiles, they are described in terms of statistical distributions of the following random variables (see SP-1 D1.2):

- Inter-packet time
- Packet size
- Inter-burst time
- Burst size

Each random variable is defined then, as a probability distribution with some parameters, e.g., mean and standard deviation for the normal distribution, the units and a scale factor, usually 1. The following table reproduces an example for a video service.

```
"video":{
    "interBurstRate": {"dist": "normal", "params": {"mean":0.25, "std":2.54e-5},
                       "unit":"s-1", "scale":1},
    "burstSize": {"dist": "normal", "params": {"mean":4034485, "std":1266989},
                  "unit": "bytes", "scale":1},
    "packetSize": { "dist":"constant", "params": {"mean":1500}, "unit": "bytes", "scale":1},
    "interPacketTime": {"dist": "normal", "params": {"mean":0.0001, "std":0.00002},
                        "unit": "s", "scale": 1}
}
```

Regarding traffic patterns, they are used to described the temporal behavior of a given service. They define, the periodicity and the specific pattern as values in the continuous range [0-1].

```
"daily": {"period": 86400, "unit": "s", "pattern": [0.25, 0.3, 0.32, 0.35, 0.39, .....],
          "pattern_unit": "min"},
"periodical": {"period": 5, "unit": "ms", "pattern": [1,0,0,0,0], "pattern_unit": "ms"}
```

Finally, traffic profiles are defined as a mix of services with different traffic patterns and scale factors. The following table shows an example of traffic profile.

```
"trafficMix": {
        "video": {"scale": 100, "pattern": "daily-1"},
        "gaming": {"scale": 300, "pattern": "daily-2"},
        "internet": {"scale": 500, "pattern": "daily-3"}
}
```

2.6.3 REST API Interface

The current version of the DT includes a REST API which implements a preliminary interface. Specifically, this version of the REST API offers several endpoints that provide methods to interact with the Topology DB and to issue the evaluation of the KPI of a selected flow. The REST API offers the following endpoints:

- NetworkDB: Allows GET, POST and DELETE methods. Implements the creation, deletion and retrieval of the Topology DB in the Digital Twin.
- Topology/Node: Allows GET, POST and DELETE methods. Implements the creation, deletion and retrieval of the Node's information in the Topology DB.
- Topology/Link: Allows POST and DELETE methods. Implements the creation and deletion of the Link's information in the Topology DB.
- Flow: Allows GET, POST and DELETE methods. Implements the creation, deletion and deletion of the Flow's information in the Topology DB.
- KPIEvaluation: Allows GET method. Implements the KPI evaluation of a selected flow of the Topology DB.

The following table describes the end-points and operations and provides examples.

Operation	Description and examples
/NETWORKDB	
CREATE TOPO DB	Initializes a Network Database with the following elements: • List of nodes: packet switches that forward the traffic • List of hosts: hosts are source and destination of traffic flows • List of links: a link connects two ports from a node or a host. • Flows: traffic flows between two hosts. Example of Request: {"url": "http://127.0.0.1:8080/NetworkDB/", "method":"POST", "body": ""}
GET TOPO DB	Returns the contents of the Topo DB. Example of Request: {"url": "http://127.0.0.1:8080/NetworkDB/", "method":"GET", "body": ""}
DELETE TOPO DB	Returns the contents of the Topo DB. Example of Request: {"url": "http://127.0.0.1:8080/NetworkDB/", "method":"GET", "body": ""}
/TOPOLOGY/NODE	

ADD HOST	Adds a host to the Network DB. All ports must be of the type “Host”.
	Example of Request:
	{"url": "http://127.0.0.1:8080/Topology/Node/", "method":"POST", "body": {
	"DPId": "H1", "data": {
	"type": "Host", "ports": {"E1": {"type": "Host", "speed": 100}}}}
	}

ADD SWITCH	Adds a switch to the Network DB. Ports can be the type “Access” to connect to a host, or type “Trunk” to connect to another switch.
	Example of Request:
	{"url": "http://127.0.0.1:8080/Topology/Node/", "method":"POST", "body": {
	"DPId": "R1", "data": {
	"type": "Switch", "ports": {
	"A1": {"type": "Access", "speed": 100},
	"A2": {"type": "Access", "speed": 100},
	"T1": {"type": "Trunk","speed": 100, "transmissionDelay": 50},
	"T2": {"type": "Trunk","speed": 100, "transmissionDelay": 50}
	}}}}

GET HOST / SWITCH	Returns the properties of a Host or a Switch. Example of Request: {"url": "http://127.0.0.1:8080/Topology/Node/", "method":"GET", "body": { "DPId": "H_FluE_1" }}
DELETE HOST / SWITCH	Removes a Host or a Switch from the Network DB. Example of Request: {"url": "http://127.0.0.1:8080/Topology/Node/", "method":"DELETE", "body": { "DPId": "H_FluE_1" }}
/TOPOLOGY/LINK	
ADD LINK	Adds a link to the Network DB. The link connects a host to a node or two switches. Examples of Request:

	Host to Switch: {"url": "http://127.0.0.1:8080/Topology/Link/", "method":"POST", "body": { "DPA": "H1", "PortA": "E1", "DPB": "R1", "PortB": "A1" }} Switch to Switch: {"url": "http://127.0.0.1:8080/Topology/Link/", "method":"POST", "body": { "DPA": "R1", "PortA": "T1", "DPB": "R3", "PortB": "T1" }}
DELETE LINK	Removes a Link from the Network DB. Example of Request: {"url": "http://127.0.0.1:8080/Topology/Link/", "method":"DELETE", "body": { "DPA": "H_FluE_1", "PortA": "E1", "DPB": "R1", "PortB": "A2" }}
/FLOW	

ADD FLOW	Adds a traffic flow to the Network DB. The traffic flow specifies the end hosts/ports, as well as the traffic of the flow.
	Example of Request:
	{"url": "http://127.0.0.1:8080/Flow/", "method":"POST", "body": {
	"flowId": "BG_Traffic1", "data": {
	"src": "H1", "srcPort": "E1", "dest": "H3", "destPort": "E1",
	"srcSwitch": "R1", "destSwitch": "R3", "path": ["R1","R3"],
	"trafficMix": {
	"video": {"scale": 100, "pattern": "daily-1"},
	"gaming": {"scale": 300, "pattern": "daily-2"},
	"internet": {"scale": 500, "pattern": "daily-3"}
	}}}}

GET FLOW	Returns the properties of a traffic flow. Example of Request: {"url": "http://127.0.0.1:8080/Flow/", "method":"GET", "body": {"flowId": "BG_Traffic1"}}
DELETE FLOW	Removes a flow from the Network DB. Example of Request: {"url": "http://127.0.0.1:8080/Flow/", "method":"DELETE", "body": { "flowId": "BG_Traffic1" }}
/KPIEVALUATION	

GET ESTIMATION	Evaluates the KPIs (delay, jitter and loss) of a new traffic flow to be established on a defined path. The KPI estimation of the new flow is for a defined traffic and takes into account the other traffic flows already established.
	Example of Request:
	{"url": "http://127.0.0.1:8080/KPIEvaluation/", "method":"GET", "body": {
	"flowId": "FluE", "data": {
	"src": "H_FluE_1", "srcPort": "E1", "dest": "H_FluE_2", "destPort": "E1",
	"srcSwitch": "R1", "destSwitch": "R3",
	"pathToEvaluate": ["R1","R3"],
	"trafficMix": {
	"video": {"scale": 28, "pattern": "daily-1"}
	}}}}

2.6.4 Report on Current Status

A preliminary REST API is already implemented and fully functional. In addition, this version of the DT includes preliminary algorithms to compute the delay in a set of queues that support a traffic flow. Initial versions of traffic definitions are also included.

2.6.5 Stand-alone preliminary tests

We have developed a REST API client that performs requests to the digital twin. The client can be configured by defining a JSON file with the requests to be submitted through the REST API interface and prints the results. Specifically, a JSON file named *Tests.json* has been defined that follows the following sequence:

- Create a network topology with 4 Hosts and 3 Switches in a ring.
- Add two flows as background traffic
- Request for RPI evaluation for a new flow

The results of the test are listed in the next table.

```
$ /usr/bin/python3.10 ./client.py -c config/Tests.json
INFO: Running REST API client...
INFO: ----------CREATE TOPO DB----------
INFO: POST http://127.0.0.1:8080/NetworkDB/
INFO: -- 201 Created - {'networkDB': {'nodeList': [], 'hostList': [], 'linkList': [], 'nodes': {}, 'flows': {}}}
INFO: ----------HOSTs----------
INFO: POST http://127.0.0.1:8080/Topology/Node/ {'DPId': 'H1', 'data': {'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}}}
INFO: -- 201 Created - {'H1': "{'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}, 'DPId': 'H1'}", 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Node/ {'DPId': 'H2', 'data': {'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}}}
INFO: -- 201 Created - {'H2': "{'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}, 'DPId': 'H2'}", 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Node/ {'DPId': 'H3', 'data': {'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}}}
INFO: -- 201 Created - {'H3': "{'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}, 'DPId': 'H3'}", 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Node/ {'DPId': 'H4', 'data': {'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}}}
INFO: -- 201 Created - {'H4': "{'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}, 'DPId': 'H4'}", 'result': 'OK'}
INFO: ----------SWITCHES----------
INFO: POST http://127.0.0.1:8080/Topology/Node/ {'DPId': 'R1', 'data': {'type': 'Switch', 'ports': {'A1': {'type': 'Access', 'speed': 100}, 'A2': {'type': 'Access', 'speed': 100}, 'T1': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}, 'T2': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}}}}
INFO: -- 201 Created - {'R1': "{'type': 'Switch', 'ports': {'A1': {'type': 'Access', 'speed': 100}, 'A2': {'type': 'Access', 'speed': 100}, 'T1': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}, 'T2': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}}, 'DPId': 'R1'}", 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Node/ {'DPId': 'R2', 'data': {'type': 'Switch', 'ports': {'A1': {'type': 'Access', 'speed': 100}, 'T1': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}, 'T2': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}}}}
INFO: -- 201 Created - {'R2': "{'type': 'Switch', 'ports': {'A1': {'type': 'Access', 'speed': 100}, 'T1': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}, 'T2': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}}, 'DPId': 'R2'}", 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Node/ {'DPId': 'R3', 'data': {'type': 'Switch', 'ports': {'A1': {'type': 'Access', 'speed': 100}, 'A2': {'type': 'Access', 'speed': 100}, 'A3': {'type': 'Access', 'speed': 100}, 'T1': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}, 'T2': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}}}}
INFO: -- 201 Created - {'R3': "{'type': 'Switch', 'ports': {'A1': {'type': 'Access', 'speed': 100}, 'A2': {'type': 'Access', 'speed': 100}, 'A3': {'type': 'Access', 'speed': 100}, 'T1': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}, 'T2': {'type': 'Trunk', 'speed': 100, 'transmissionDelay': 50}}, 'DPId': 'R3'}", 'result': 'OK'}
INFO: ----------ACCESS LINKS----------
INFO: POST http://127.0.0.1:8080/Topology/Link/ {'DPA': 'H1', 'PortA': 'E1', 'DPB': 'R1', 'PortB': 'A1'}
```

INFO: -- 200 OK - {'link': {'DPA': 'H1', 'PortA': 'E1', 'DPB': 'R1', 'PortB': 'A1'}, 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Link/ {'DPA': 'H2', 'PortA': 'E1', 'DPB': 'R2', 'PortB': 'A1'}
INFO: -- 200 OK - {'link': {'DPA': 'H2', 'PortA': 'E1', 'DPB': 'R2', 'PortB': 'A1'}, 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Link/ {'DPA': 'R3', 'PortA': 'A1', 'DPB': 'H3', 'PortB': 'E1'}
INFO: -- 200 OK - {'link': {'DPA': 'R3', 'PortA': 'A1', 'DPB': 'H3', 'PortB': 'E1'}, 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Link/ {'DPA': 'R3', 'PortA': 'A2', 'DPB': 'H4', 'PortB': 'E1'}
INFO: -- 200 OK - {'link': {'DPA': 'R3', 'PortA': 'A2', 'DPB': 'H4', 'PortB': 'E1'}, 'result': 'OK'}
INFO: ----------TOPOLOGY LINKS----------
INFO: POST http://127.0.0.1:8080/Topology/Link/ {'DPA': 'R1', 'PortA': 'T1', 'DPB': 'R3', 'PortB': 'T1'}
INFO: -- 200 OK - {'link': {'DPA': 'R1', 'PortA': 'T1', 'DPB': 'R3', 'PortB': 'T1'}, 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Link/ {'DPA': 'R1', 'PortA': 'T2', 'DPB': 'R2', 'PortB': 'T1'}
INFO: -- 200 OK - {'link': {'DPA': 'R1', 'PortA': 'T2', 'DPB': 'R2', 'PortB': 'T1'}, 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Link/ {'DPA': 'R2', 'PortA': 'T2', 'DPB': 'R3', 'PortB': 'T2'}
INFO: -- 200 OK - {'link': {'DPA': 'R2', 'PortA': 'T2', 'DPB': 'R3', 'PortB': 'T2'}, 'result': 'OK'}
INFO: ----------FLOWS----------
INFO: POST http://127.0.0.1:8080/Flow/ {'flowId': 'BG_Traffic1', 'data': {'src': 'H1', 'srcPort': 'E1', 'dest': 'H3', 'destPort': 'E1', 'srcSwitch': 'R1', 'destSwitch': 'R3', 'path': ['R1', 'R3'], 'trafficMix': {'video': {'scale': 100, 'pattern': 'daily-1'}, 'gaming': {'scale': 300, 'pattern': 'daily-2'}, 'internet': {'scale': 500, 'pattern': 'daily-3'}}}}
INFO: -- 201 Created - {'BG_Traffic1': "{'src': 'H1', 'srcPort': 'E1', 'dest': 'H3', 'destPort': 'E1', 'srcSwitch': 'R1', 'destSwitch': 'R3', 'path': ['R1', 'R3'], 'trafficMix': {'video': {'scale': 100, 'pattern': 'daily-1'}, 'gaming': {'scale': 300, 'pattern': 'daily-2'}, 'internet': {'scale': 500, 'pattern': 'daily-3'}}, 'flowId': 'BG_Traffic1'}", 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Flow/ {'flowId': 'BG_Traffic2', 'data': {'src': 'H2', 'srcPort': 'E1', 'dest': 'H4', 'destPort': 'E1', 'srcSwitch': 'R2', 'destSwitch': 'R3', 'path': ['R2', 'R3'], 'trafficMix': {'video': {'scale': 100, 'pattern': 'daily-1'}, 'gaming': {'scale': 300, 'pattern': 'daily-2'}, 'internet': {'scale': 500, 'pattern': 'daily-3'}}}}
INFO: -- 201 Created - {'BG_Traffic2': "{'src': 'H2', 'srcPort': 'E1', 'dest': 'H4', 'destPort': 'E1', 'srcSwitch': 'R2', 'destSwitch': 'R3', 'path': ['R2', 'R3'], 'trafficMix': {'video': {'scale': 100, 'pattern': 'daily-1'}, 'gaming': {'scale': 300, 'pattern': 'daily-2'}, 'internet': {'scale': 500, 'pattern': 'daily-3'}}, 'flowId': 'BG_Traffic2'}", 'result': 'OK'}
INFO: ----------GETs----------
INFO: GET http://127.0.0.1:8080/NetworkDB/
INFO: -- 200 OK - {'nodeList': ['R1', 'R2', 'R3'], 'hostList': ['H1', 'H2', 'H3', 'H4'], 'linkList': [{'DPA': 'H1', 'PortA': 'E1', 'DPB': 'R1', 'PortB': 'A1'}, {'DPA': 'H2', 'PortA': 'E1', 'DPB': 'R2', 'PortB': 'A1'}, {'DPA': 'R3', 'PortA': 'A1', 'DPB': 'H3', 'PortB': 'E1'}, {'DPA': 'R3', 'PortA': 'A2', 'DPB': 'H4', 'PortB': 'E1'}, {'DPA': 'R1', 'PortA': 'T1', 'DPB': 'R3', 'PortB': 'T1'}, {'DPA': 'R1', 'PortA': 'T2', 'DPB': 'R2', 'PortB': 'T1'}, {'DPA': 'R2', 'PortA': 'T2', 'DPB': 'R3', 'PortB': 'T2'}], 'nodes': {'H1': {'DPId': 'H1', 'type': 'Host', 'ports': {'E1': {'DPId': 'H1', 'portId': 'E1', 'speed': 100, 'type': 'Host', 'neighbour': {'DPId': 'R1', 'portId': 'A1'}, 'inFlows': [], 'outFlows': ['BG_Traffic1']}}, 'Neighbours2Ports': {'R1': 'E1'}, 'flows': ['BG_Traffic1']}, 'H2': {'DPId': 'H2', 'type': 'Host', 'ports': {'E1': {'DPId': 'H2', 'portId': 'E1', 'speed': 100, 'type': 'Host', 'neighbour': {'DPId': 'R2', 'portId': 'A1'}, 'inFlows': [], 'outFlows': ['BG_Traffic2']}}, 'Neighbours2Ports': {'R2': 'E1'}, 'flows': ['BG_Traffic2']}, 'H3': {'DPId': 'H3', 'type': 'Host', 'ports': {'E1': {'DPId': 'H3', 'portId': 'E1', 'speed': 100, 'type': 'Host', 'neighbour': {'DPId': 'R3', 'portId': 'A1'}, 'inFlows': ['BG_Traffic1'], 'outFlows': []}}, 'Neighbours2Ports': {'R3': 'E1'}, 'flows': ['BG_Traffic1']}, 'H4': {'DPId': 'H4', 'type': 'Host', 'ports': {'E1': {'DPId': 'H4', 'portId': 'E1', 'speed': 100, 'type': 'Host', 'neighbour': {'DPId': 'R3', 'portId': 'A2'}, 'inFlows': ['BG_Traffic2'], 'outFlows': []}}, 'Neighbours2Ports': {'R3': 'E1'}, 'flows': ['BG_Traffic2']}, 'R1': {'DPId': 'R1', 'type': 'Switch', 'ports': {'A1': {'DPId': 'R1', 'portId': 'A1', 'speed': 100, 'type': 'Access', 'neighbour': {'DPId': 'H1', 'portId': 'E1'}, 'inFlows': ['BG_Traffic1'], 'outFlows': []}, 'A2': {'DPId': 'R1', 'portId': 'A2', 'speed': 100, 'type': 'Access', 'neighbour': None, 'inFlows': [], 'outFlows': []}, 'T1': {'DPId': 'R1', 'portId': 'T1', 'speed': 100, 'type': 'Trunk', 'neighbour': {'DPId': 'R3', 'portId': 'T1'}, 'inFlows': [], 'outFlows': ['BG_Traffic1'], 'transmissionDelay': 50.0}, 'T2': {'DPId': 'R1', 'portId': 'T2', 'speed': 100, 'type': 'Trunk', 'neighbour': {'DPId': 'R2', 'portId': 'T1'}, 'inFlows': [], 'outFlows': [], 'transmissionDelay': 50.0}}, 'Neighbours2Ports': {'H1': 'A1', 'R3': 'T1', 'R2': 'T2'}, 'flows': ['BG_Traffic1']}, 'R2': {'DPId': 'R2', 'type': 'Switch', 'ports': {'A1': {'DPId': 'R2', 'portId': 'A1', 'speed': 100, 'type': 'Access', 'neighbour': {'DPId': 'H2', 'portId': 'E1'}, 'inFlows': ['BG_Traffic2'], 'outFlows': []}, 'T1': {'DPId': 'R2', 'portId': 'T1', 'speed': 100, 'type': 'Trunk', 'neighbour': {'DPId': 'R1', 'portId': 'T2'}, 'inFlows': [], 'outFlows': [], 'transmissionDelay': 50.0}, 'T2': {'DPId': 'R2', 'portId': 'T2', 'speed': 100, 'type': 'Trunk', 'neighbour': {'DPId': 'R3', 'portId': 'T2'}, 'inFlows': [], 'outFlows': ['BG_Traffic2'], 'transmissionDelay': 50.0}}, 'Neighbours2Ports': {'H2': 'A1', 'R1': 'T1', 'R3': 'T2'}, 'flows': ['BG_Traffic2']}, 'R3': {'DPId': 'R3', 'type': 'Switch', 'ports': {'A1': {'DPId': 'R3', 'portId': 'A1', 'speed': 100, 'type': 'Access', 'neighbour': {'DPId': 'H3', 'portId': 'E1'}, 'inFlows': [], 'outFlows': ['BG_Traffic1']}, 'A2': {'DPId': 'R3', 'portId': 'A2', 'speed': 100, 'type': 'Access', 'neighbour': {'DPId': 'H4', 'portId': 'E1'}, 'inFlows': [], 'outFlows': ['BG_Traffic2']}, 'A3': {'DPId': 'R3', 'portId': 'A3', 'speed': 100, 'type': 'Access', 'neighbour': None, 'inFlows': [], 'outFlows': []}, 'T1': {'DPId': 'R3', 'portId': 'T1', 'speed': 100, 'type': 'Trunk', 'neighbour': {'DPId': 'R1', 'portId': 'T1'}, 'inFlows': ['BG_Traffic1'], 'outFlows': [], 'transmissionDelay': 50.0}, 'T2': {'DPId': 'R3', 'portId': 'T2', 'speed': 100, 'type': 'Trunk', 'neighbour': {'DPId': 'R2', 'portId': 'T2'}, 'inFlows': ['BG_Traffic2'], 'outFlows': [], 'transmissionDelay': 50.0}}, 'Neighbours2Ports': {'H3': 'A1', 'H4': 'A2', 'R1': 'T1', 'R2': 'T2'}, 'flows': ['BG_Traffic2',

'BG_Traffic1']}}, 'flows': {'BG_Traffic1': {'flowId': 'BG_Traffic1', 'src': 'H1', 'dest': 'H3', 'srcSwitch': 'R1', 'destSwitch': 'R3', 'srcPort': 'E1', 'destPort': 'E1', 'path': ['R1', 'R3'], 'trafficMix': {'video': {'scale': 100, 'pattern': 'daily-1'}, 'gaming': {'scale': 300, 'pattern': 'daily-2'}, 'internet': {'scale': 500, 'pattern': 'daily-3'}}}, 'BG_Traffic2': {'flowId': 'BG_Traffic2', 'src': 'H2', 'dest': 'H4', 'srcSwitch': 'R2', 'destSwitch': 'R3', 'srcPort': 'E1', 'destPort': 'E1', 'path': ['R2', 'R3'], 'trafficMix': {'video': {'scale': 100, 'pattern': 'daily-1'}, 'gaming': {'scale': 300, 'pattern': 'daily-2'}, 'internet': {'scale': 500, 'pattern': 'daily-3'}}}}}
INFO: GET http://127.0.0.1:8080/Flow/ {'flowId': 'BG_Traffic2'}
INFO: -- 200 OK - {'BG_Traffic2': {'flowId': 'BG_Traffic2', 'src': 'H2', 'dest': 'H4', 'srcSwitch': 'R2', 'destSwitch': 'R3', 'srcPort': 'E1', 'destPort': 'E1', 'path': ['R2', 'R3'], 'trafficMix': {'video': {'scale': 100, 'pattern': 'daily-1'}, 'gaming': {'scale': 300, 'pattern': 'daily-2'}, 'internet': {'scale': 500, 'pattern': 'daily-3'}}}}
INFO: ----------KPI EVALUATION----------
INFO: POST http://127.0.0.1:8080/Topology/Node/ {'DPId': 'H_FluE_1', 'data': {'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}}}
INFO: -- 201 Created - {'H_FluE_1': "{'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}, 'DPId': 'H_FluE_1'}", 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Node/ {'DPId': 'H_FluE_2', 'data': {'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}}}
INFO: -- 201 Created - {'H_FluE_2': "{'type': 'Host', 'ports': {'E1': {'type': 'Host', 'speed': 100}}, 'DPId': 'H_FluE_2'}", 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Link/ {'DPA': 'H_FluE_1', 'PortA': 'E1', 'DPB': 'R1', 'PortB': 'A2'}
INFO: -- 200 OK - {'link': {'DPA': 'H_FluE_1', 'PortA': 'E1', 'DPB': 'R1', 'PortB': 'A2'}, 'result': 'OK'}
INFO: POST http://127.0.0.1:8080/Topology/Link/ {'DPA': 'R3', 'PortA': 'A3', 'DPB': 'H_FluE_2', 'PortB': 'E1'}
INFO: -- 200 OK - {'link': {'DPA': 'R3', 'PortA': 'A3', 'DPB': 'H_FluE_2', 'PortB': 'E1'}, 'result': 'OK'}
INFO: GET http://127.0.0.1:8080/KPIEvaluation/ {'flowId': 'FluE', 'data': {'src': 'H_FluE_1', 'srcPort': 'E1', 'dest': 'H_FluE_2', 'destPort': 'E1', 'srcSwitch': 'R1', 'destSwitch': 'R3', 'pathToEvaluate': ['R1', 'R3'], 'trafficMix': {'video': {'scale': 28, 'pattern': 'daily-1'}}}}
INFO: -- 200 OK - {'flowId': 'FluE', 'kpis': {'delay_us': 538, 'jitter_us': 35, 'loss': 0}, 'result': 'OK'}
INFO: DELETE http://127.0.0.1:8080/Topology/Link/ {'DPA': 'H_FluE_1', 'PortA': 'E1', 'DPB': 'R1', 'PortB': 'A2'}
INFO: -- 200 OK - {'link': {'DPA': 'H_FluE_1', 'PortA': 'E1', 'DPB': 'R1', 'PortB': 'A2'}, 'result': 'OK'}
INFO: DELETE http://127.0.0.1:8080/Topology/Link/ {'DPA': 'R3', 'PortA': 'A3', 'DPB': 'H_FluE_2', 'PortB': 'E1'}
INFO: -- 200 OK - {'link': {'DPA': 'R3', 'PortA': 'A3', 'DPB': 'H_FluE_2', 'PortB': 'E1'}, 'result': 'OK'}
INFO: DELETE http://127.0.0.1:8080/Topology/Node/ {'DPId': 'H_FluE_1'}
INFO: -- 200 OK - {'delete': 'H_FluE_1', 'result': 'OK'}
INFO: DELETE http://127.0.0.1:8080/Topology/Node/ {'DPId': 'H_FluE_2'}
INFO: -- 200 OK - {'delete': 'H_FluE_2', 'result': 'OK'}
INFO: REST API client finished

2.7 METRO SDN CONTROLLER

The TSN domains are expected to be interconnected by means of a Metro Area Network, based on packet-switching technology, i.e., and IP network.

Two aspects should be considered:

- The specific network nodes forming the packet-switching infrastructure.
- The SDN controller used to configure the paths tailored to the characteristics required for satisfying the interconnection of the TSN domains. The goal is to leverage on the concept of network slicing so that service guarantees can be provided to certain flows.

At the time of designing the use case, the initial approach was the creation of an IP network infrastructure for interconnecting the TSN domains, being this IP infrastructure being controlled by an SDN Controller. While being realistic, this option represents a challenge from the point of view of the alignment of existing solutions, for both control and data plane, with the support of the network slicing concept being currently defined in IETF. Note that the purpose of the scenario is to validate the effectiveness of the transport slice approach, rather to resolve the integration issues of current state of the art.

Thus, a new strategy is being analzed, as work in progress to complete the scenario. Such strategy is as follows.

Taking advantage of the fact that the TSN switches in the setup support as much as 1 Gbps bit rate interfaces, a way of emulating the behavior of an IP network from the perspective of the data plane is to use instrumentation devices able to introduce impairments on the network links. That is, apply a given behavior in terms of bandwidth, latency, jitter and packet loss to the interfaces of interest. In this manner, different behaviors can be applied to different flows differentiated e.g. by vlans. This is sufficient to emulate the behavior of network slices in the Metro network.

On the other hand, the control elements of TIMING need to interact with the emulated IP network, such interaction following the network slice model as defined in draft-ietf-teas-ietf-network-slice-nbi-yang. To that end, the purpose is to develop a control module able to handle the networks slicing NBI in its interaction with the control elements of TIMING, and program the instrumentation equipment for the applications of specific impairmenta to the flows received from the TSN domains.

As said, the interaction with the control elements is expected to follow the IETF NBI YANG model for slicing, so the same YANG model is assumed to be supported. For the interaction with the instrumentation vendor, the idea is to leverage on a programmable API from the instrumentation device.

This is a work in progress, not yet prototyped.

2.8 INDUSTRIAL APPLICATIONS

2.8.1 Description

The architecture of the AGV control software is shown in Figure 2-13.

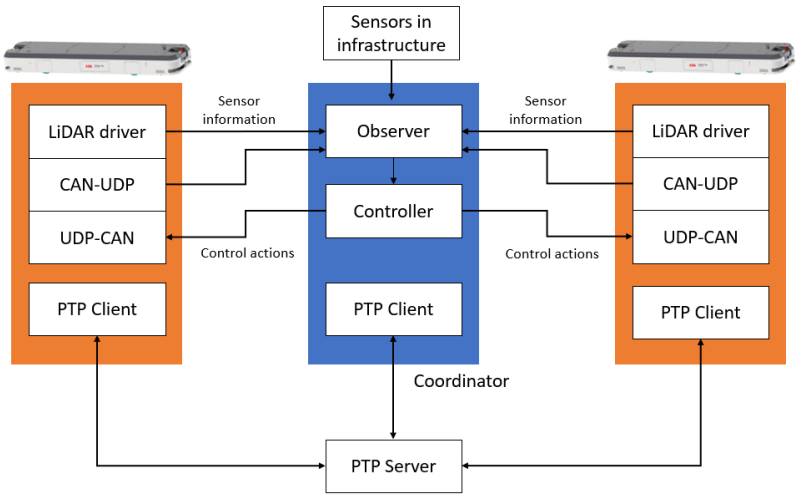


Figure 2-13: AGV control architecture

Each AGV is equipped with an embedded computer to run all software components based on Linux. This hardware is connected to the AGV by CAN bus. Each AGV sends the information of

its sensors to the coordinator. To do it, inside each AGV there is a component that is connected to the CAN bus and filters the CAN frames with the sensor information. In addition, each AGV is equipped with a LiDAR. The LiDAR driver receives the point cloud information and send it to the coordinator. The coordinator considering the sensor information, generates the control actions needed to govern the movement of the AGVs. In this control problem, the control actions are the longitudinal speed and the steering angle. Finally, there is a PTP inside each AGV to synchronize the internal time by a PTP server.

2.8.1.1 Coordinator

The coordinator is composed by an observer and a controller (Figure 2-14).

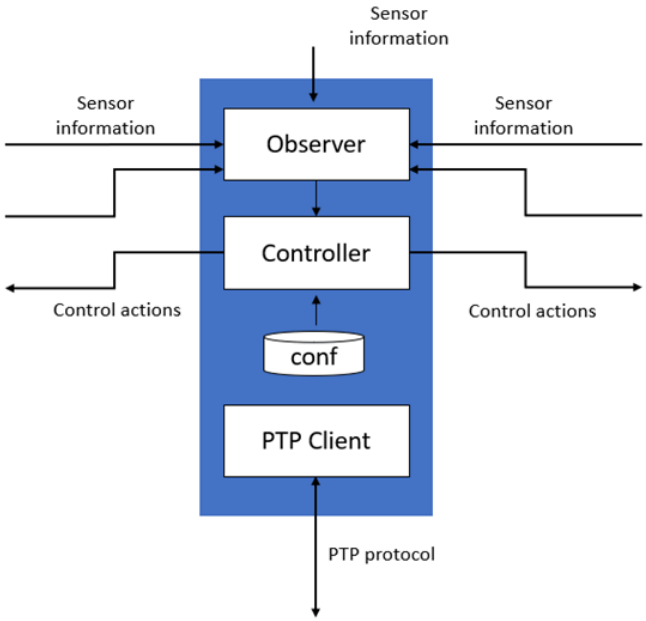


Figure 2-14: AGV coordinator SW modules

The observer receives the sensor information (speed, and point cloud from the LiDARs) and estimates the current position of each AGV and its speed $(v_1, w_1, x_1, y_1, \theta_1, v_2, w_2, x_2, y_2, \theta_2)$. This information is provided to the controller.

The controller uses this information to generate the speed and steering commands for the AGVs $(v_{r_1}, \gamma_1, v_{r_2}, \gamma_2)$ required to follow the speed profile stablished in the configuration file.

2.8.1.2 AGV software

The AGV is equipped with an external hardware to run the software components. This hardware is connected to the AGV by a CAN bus. The software components in this hardware are shown in Figure 2-15.

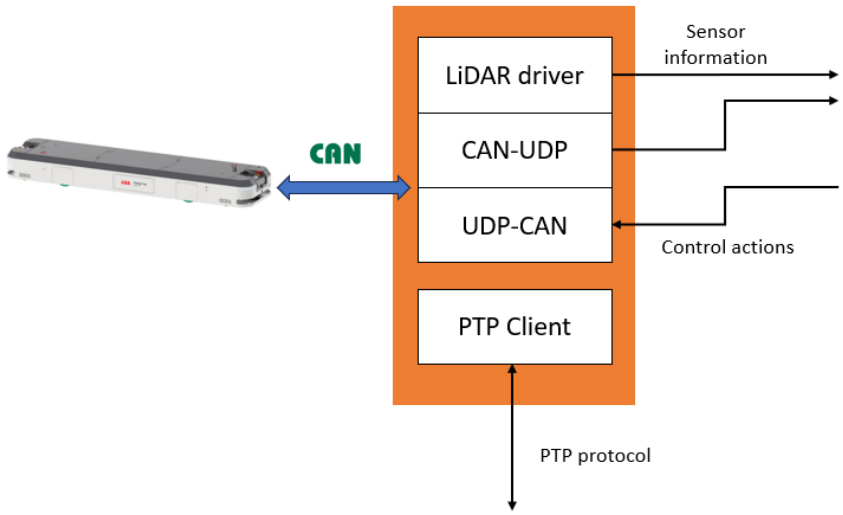


Figure 2-15: AGV software modules

A component manages the communication with the LiDAR and acts as a driver sending this information by UDP. Another component listens the CAN frames, filter the frames related with the sensors in the AGV, and send this information by UDP. A component receives the control actions form UDP frames and translate them to CAN frames. In addition, a PTP client manages the time synchronization in the system.

2.8.2 Report on the Current Status

The following modules have been (or are being) developed/implemented:

- **Observer (OBS)**: It will be implemented in python. It requires UDP communication (library socket).
- **Controller (ACON)**: It will be implemented in python. It requires UDP communication (library socket).
- **CAN-UDP**: Implemented in python. It requires UDP communication (library socket). It requires CAN communication (library python-can). Tested without TSN. Used to capture traffic.
- **UDP-CAN**: Implemented in python. It requires UDP communication (library socket). It requires CAN communication (library python-can). Tested without TSN. Testing with TSN pending.
- **LiDAR driver**: An initial version has been implemented in C#. A new version will be implemented in python or C++.It requires TCP and UDP communication (library socket). Tested without TSN. Used to capture traffic. Testing with TSN pending.
- **PTP client**: The software component is ready in http://git.code.sf.net/p/linuxptp/code. Testing pending of BeagleBone reception.

In the following tables, the details of the preliminary tests that have been conducted are reported. In particular, for each test, the description, the steps and the preliminary results are detailed.

Test #1

Test Id	ABBagvCon_PTPclient_timesync_1
Description	
This test validates that the time is correctly synchronized in the AGV and the coordinator	
Preconditions	
The AGV is on, and the PTP client in AGVs and coordinator are not running. The batteries for the RTC are disconnected. All elements are connected in a network with a PTP server.	
Execution Steps	
1. Connect the batteries for the RTC 2. Run the PTC client in the AGVs and the coordinator	
Expected Results	
The time are synchronized with the PTP server	
Output	
ptp4l, phc2sys should report the time offset between PHC and System Clock, which determines if the clocks are synchronized Example: phc2sys[5374168.545]: CLOCK_REALTIME phc offset -372582 s0 freq +246 delay 6649 phc2sys[5374169.545]: CLOCK_REALTIME phc offset -372832 s1 freq -4 delay 6673 phc2sys[5374170.547]: CLOCK_REALTIME phc offset 68 s2 freq +64 delay 6640 phc2sys[5374171.547]: CLOCK_REALTIME phc offset -20 s2 freq -3 delay 6687 phc2sys[5374172.547]: CLOCK_REALTIME phc offset 47 s2 freq +58 delay 6619 phc2sys[5374173.548]: CLOCK_REALTIME phc offset -40 s2 freq -15 delay 6680	
Execution	
Pending of beagle bone reception.	

Test #2	
Test Id	ABBagvCon_PTPclient_timesync_2
Description	
This test validates that the time is correctly synchronized in the AGV and the coordinator, in the event of a change in the system time. All elements are connected in a network with a PTP server.	
Preconditions	

The AGV is on, and the PTP client in AGVs and coordinator are running.
Execution Steps
1. Manually change the system time to force a difference respect the time in the server
Expected Results
The time must synchronize with the PTP server. Measure the time needed to achieve the synchronization.
Output
ptp4l, phc2sys should report the time offset between PHC and System Clock, which determines if the clocks are synchronized Example: phc2sys[5374168.545]: CLOCK_REALTIME phc offset -372582 s0 freq +246 delay 6649 phc2sys[5374169.545]: CLOCK_REALTIME phc offset -372832 s1 freq -4 delay 6673 phc2sys[5374170.547]: CLOCK_REALTIME phc offset 68 s2 freq +64 delay 6640 phc2sys[5374171.547]: CLOCK_REALTIME phc offset -20 s2 freq -3 delay 6687 phc2sys[5374172.547]: CLOCK_REALTIME phc offset 47 s2 freq +58 delay 6619 phc2sys[5374173.548]: CLOCK_REALTIME phc offset -40 s2 freq -15 delay 6680
Execution
Pending of beagle bone reception

Test #3	
Test Id	ABBagvCon_OBS_CANinf_1
Description	
This test validates that the CAN frames are correctly processed	
Preconditions	
The AGVs are ON, coordinator is ON, they are in the same local network. CAN-UDP is running, Observer is running.	
Execution Steps	
1. Set a speed 0 in AGV 1. 2. Check the output of OBS 3. Set a speed 0 in AGV 2. 4. Check the output of OBS 5. Set a speed 0.3 in AGV 1.	

6. Check the output of OBS 7. Set a speed 0.3 in AGV 2. 8. Check the output of OBS
Expected Results
The observer should correctly estimate the profile speed of the AGVs.
Output
OBS generates a file with the speed estimated from the AGVs.
Execution
CAN-UDP correctly sends the information, processing in observer is pending.

Test #4	
Test Id	ABBagvCon_OBS_LIDinf_1
Description	
This test validates that the LiDAR information is correctly processed	
Preconditions	
The AGVs are ON, coordinator is ON, they are in the same local network. CAN-UDP is not running, Observer is running. One LiDAR is connected to each AGV. AGVs are stopped. LiDARs are seeing a rectangular environment of predefined dimensions.	
Execution Steps	
1. Delete the log of OBS 2. Check the output of OBS. The position estimated of the AGVs should be match with the real one 3. Move AGV 1 4. Check the output of OBS. The position estimated of the AGVs should be match with the real one 5. Move AGV 2 6. Check the output of OBS. The position estimated of the AGVs should be match with the real one	
Expected Results	
The observer should correctly estimate the position of the AGVs	
Output	
OBS generates a file with the position estimated from the AGVs. This file will show the timestamp and the position.	

Execution
Lidar driver has been tested. Testing of observer is pending.

3 OPERATIONAL WORKFLOW FOR E2E SERVICE PROVISIONING

In this section, we present a preliminary version of the operational workflow for the provisioning of end-to-end (E2E) services across the multiple considered technological domains. The workflow will be refined as soon as the integration work advances and changes, if any, will be reported in the next deliverables. Figure 3-1 below depicts a diagram of the sequence of involved operations and actors. Namely, the E2E service provisioning entails the following operations:

1. Initially, a service provisioning request arrives to the northbound of the Connectivity Manager (CM). This request may come from a client system or an end-user, and includes the necessary details to set-up and configure the service. In particular, the request contains: the set of endpoints between which the service should be set-up; the service type; a traffic profile that characterizes the communication requirements, such as the required bandwidth; and a collection of Service Level Agreements (SLAs) that state other characteristics of the service, such as the expected degree of reliability and the bounded Key Performance Indicators (KPIs).
2. Upon reception, the CM contacts the TSN Controller so as to fetch the most updated data in regards to the infrastructure topology and capabilities. This information is necessary in order to determine which are the viable network paths from a resource availability and connectivity perspective.
3. The TSN controller replies with the information.
4. Similarly, the CM contacts the Metro controller to fetch information about the underlying Metro network infrastructure.
5. The Metro controller replies with the information.
6. With the updated information, the CM computes the candidate network paths.
7. Once computed, the CM interacts with the Digital Twin (DT) so as to assess the KPIs of the computed path, since they have to be both compliant with what the service request, and the selection of the path should not affect negatively to already established E2E services.
8. The DT employs as information to estimate the expected KPIs the requested service information and the currently active flows over the candidate paths. Once determined, the KPIs are returned to the CM.
9. With the assessed KPIs, the CM takes a final decision of which is the network path that is going to be configured across the interconnected technological domains.
10. The CM contacts with TSN controller with a Connectivity configuration request according to the decided network path, which may include nodes at the multiple technological domains, such as WiFi Access Points (AP) or wired TSN-enabled Ethernet switches.
11. The TSN controller sends the specific configuration details to the domain controller or API of the WiFi nodes included in the computed path.
12. Similar to the previous step, the TSN controller sends the specific configuration details to the domain controller or API of the Ethernet nodes included in the computed path.
13. In the same fashion, the CM contacts the Metro controller with a Connectivity configuration request so as to modify if needed any underlying configuration at the Metro network so as to satisfy the requirements of the computed path.

14. As a last step, the Metro controller applies the received configurations to the physical equipment. At this point, the E2E connectivity is fully provisioned and ready to use by upper layer services.

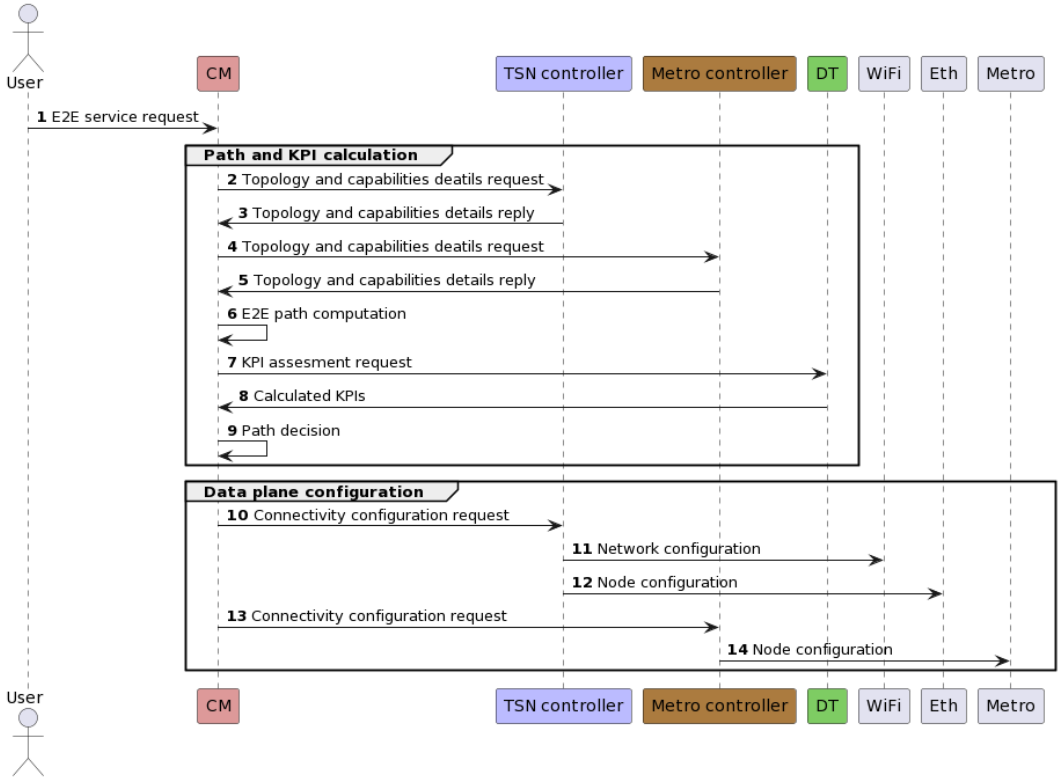


Figure 3-1: E2E service provisioning workflow.

4 PRELIMINARY INTEGRATIONS

In this section, we report some preliminary integration tests that have been already done and involving some of the TIMING architecture components. In particular, here it is reported the integration works and results of data plane devices interconnection, that is, the TSN Ethernet switch and the Wi-Fi node.

4.1 PRELIMINARY DATA PLANE INTERCONNECTION TESTS

4.1.1 PTP Synchronization tests

Precision Time Protocol (PTP) synchronization plays a pivotal role in Time-Sensitive Networking (TSN) technology as it ensures that devices within a TSN network are tightly synchronized, allowing for accurate and predictable transmission of data. Therefore, the first tests that have been carried out in the data plane interconnection have been in charge of testing the interoperability of the different PTP daemons running in the TSN WiFi Access Point and in the Ethernet TSN switch.

On this purpose, a WiFi TSN AP and an Ethernet TSN switch have been connected together via Ethernet and the synchronization precision that they achieve has been measured for 180 seconds comparing the Pulse-Per-Second (PPS) signal that both devices produce with a Keysight 53230A frequency counter. Each device produces a PPS signal in the zero-crossing of their internal timer-counter. These timer-counters are the ones that get synchronized with PTP. In the tested setup the TSN switch acts as PTP master while the WiFi-TSN AP acts as PTP slave.

Besides, the synchronization information that the PTP daemon offers in the WiFi-TSN AP has also been gathered. The PTP daemon calculates this information based on the different timestamps that it receives both from itself and from the TSN switch.

The picture and block diagram below depict the tested scenario.

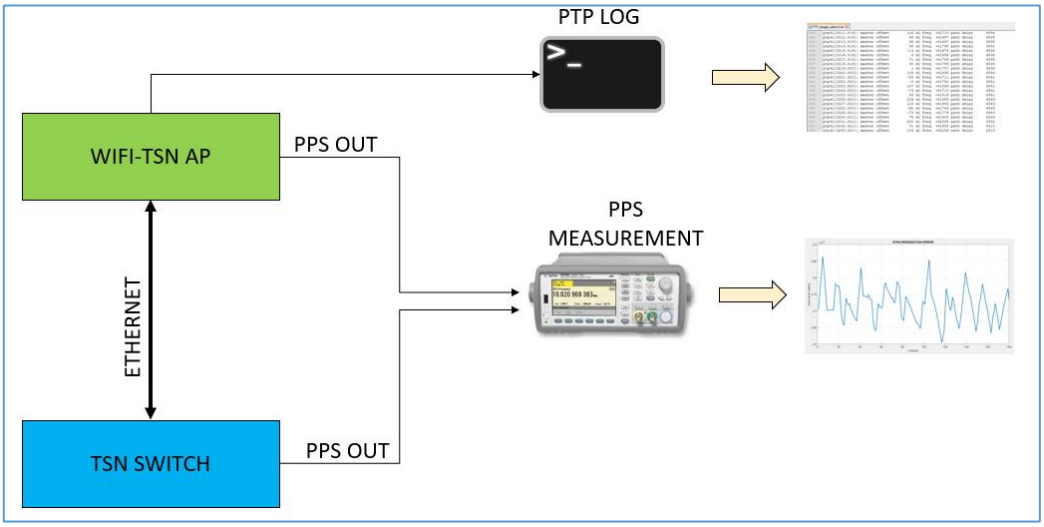


Figure 4-1: Block diagram of the PTP synchronization measurement setup

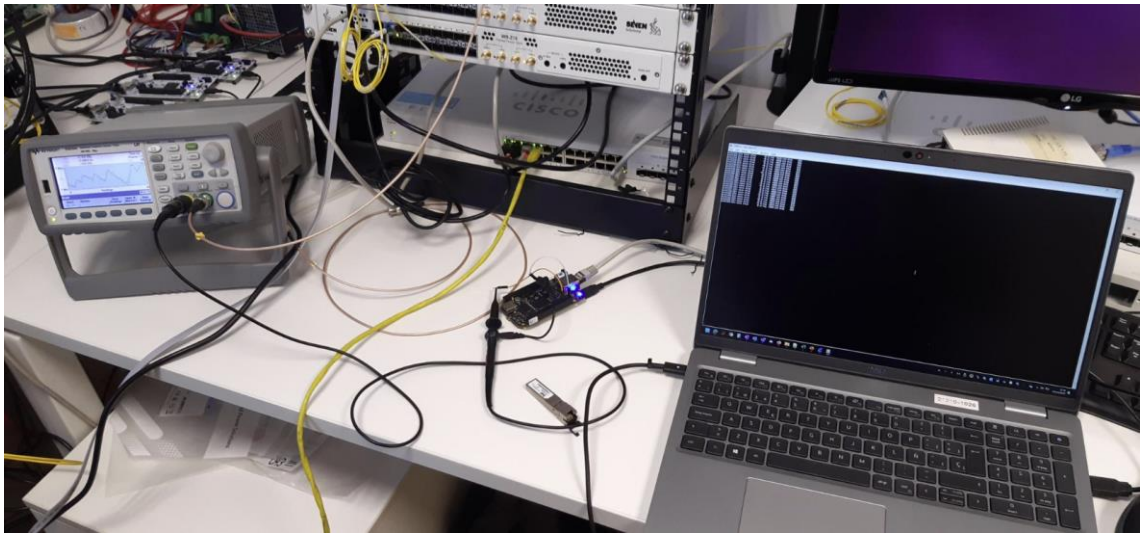


Figure 4-2: PTP synchronization measurement setup

The obtained results from the PPS measurement can be seen in the following figures:

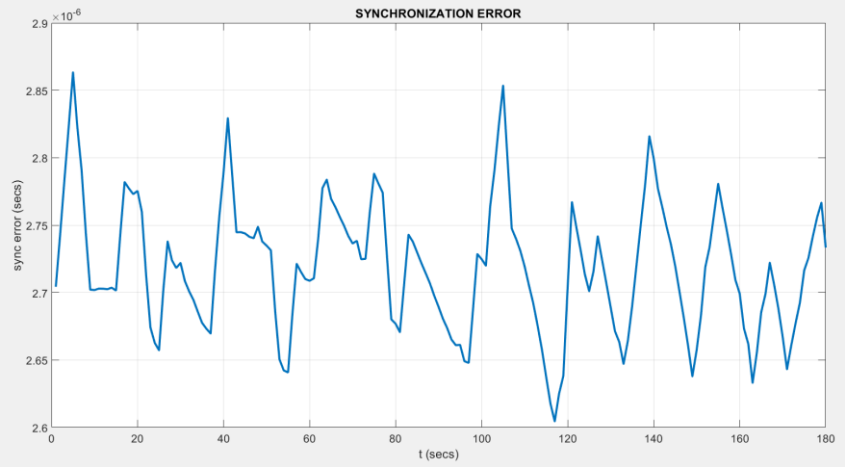


Figure 4-3: PTP synchronization error vs time

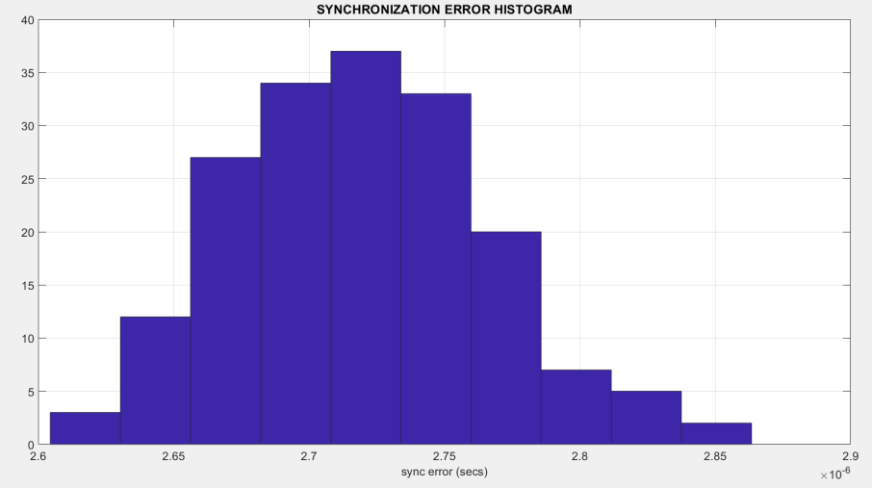


Figure 4-4: PTP synchronization error histogram

As can be observed, both devices get synchronized within a +-180 ns jitter that evolves during time. That is the expected result with a PTP synchronization. However, the mean synchronization error is expected to be around 0 while in the performed tests a 2.7 us error can be observed. This is something that needs to be further investigated.

The synchronization log that has been obtained from the PTP daemon in the WiFi-TSN AP is the following:

```
root@sharp:/home# ./Ini_PTP.sh
ptp4l[2589.106]: selected /dev/ptp0 as PTP clock
ptp4l[2589.161]: port 1: INITIALIZING to LISTENING on INITIALIZE
ptp4l[2589.164]: port 0: INITIALIZING to LISTENING on INITIALIZE
ptp4l[2589.168]: port 1: link up
ptp4l[2589.745]: port 1: received SYNC without timestamp
ptp4l[2589.749]: port 1: new foreign master 64fb81.fffe.2090c9-1
ptp4l[2591.749]: selected best master clock 64fb81.fffe.2090c9
ptp4l[2591.749]: port 1: LISTENING to UNCALIBRATED on RS_SLAVE
ptp4l[2592.745]: master offset      -3095 s0 freq  +40277 path delay      6571
ptp4l[2593.746]: master offset      -2913 s2 freq  +40459 path delay      6543
ptp4l[2593.746]: port 1: UNCALIBRATED to SLAVE on MASTER_CLOCK_SELECTED
ptp4l[2594.746]: master offset      -3217 s2 freq  +37242 path delay      6551
ptp4l[2595.746]: master offset       -174 s2 freq  +39320 path delay      6561
ptp4l[2596.746]: master offset        860 s2 freq  +40302 path delay      6571
ptp4l[2597.746]: master offset        865 s2 freq  +40565 path delay      6576
ptp4l[2598.746]: master offset        665 s2 freq  +40624 path delay      6582
ptp4l[2599.746]: master offset        340 s2 freq  +40499 path delay      6585
ptp4l[2600.746]: master offset        151 s2 freq  +40412 path delay      6585
ptp4l[2601.746]: master offset        118 s2 freq  +40424 path delay      6576
ptp4l[2602.747]: master offset        126 s2 freq  +40467 path delay      6575
ptp4l[2603.747]: master offset         14 s2 freq  +40393 path delay      6578
ptp4l[2604.747]: master offset        -42 s2 freq  +40341 path delay      6578
ptp4l[2605.747]: master offset        -42 s2 freq  +40329 path delay      6571
ptp4l[2606.747]: master offset         59 s2 freq  +40417 path delay      6560
ptp4l[2607.747]: master offset        138 s2 freq  +40514 path delay      6551
ptp4l[2608.747]: master offset        -68 s2 freq  +40349 path delay      6541
ptp4l[2609.747]: master offset         41 s2 freq  +40438 path delay      6529
ptp4l[2610.747]: master offset        -46 s2 freq  +40363 path delay      6529
ptp4l[2611.747]: master offset        -24 s2 freq  +40371 path delay      6541
ptp4l[2612.748]: master offset        217 s2 freq  +40605 path delay      6525
ptp4l[2613.748]: master offset        -58 s2 freq  +40395 path delay      6525
ptp4l[2614.748]: master offset         10 s2 freq  +40446 path delay      6532
ptp4l[2615.748]: master offset         29 s2 freq  +40468 path delay      6525
ptp4l[2616.748]: master offset          2 s2 freq  +40450 path delay      6527
ptp4l[2617.748]: master offset         21 s2 freq  +40469 path delay      6522
ptp4l[2618.748]: master offset        128 s2 freq  +40582 path delay      6528
ptp4l[2619.748]: master offset          4 s2 freq  +40497 path delay      6533
ptp4l[2620.748]: master offset         10 s2 freq  +40504 path delay      6528
ptp4l[2621.749]: master offset        -56 s2 freq  +40441 path delay      6528
ptp4l[2622.749]: master offset         20 s2 freq  +40500 path delay      6528
ptp4l[2623.749]: master offset         44 s2 freq  +40530 path delay      6528
ptp4l[2624.749]: master offset        189 s2 freq  +40688 path delay      6528
ptp4l[2625.749]: master offset        -81 s2 freq  +40475 path delay      6528
ptp4l[2626.749]: master offset        -40 s2 freq  +40492 path delay      6529
ptp4l[2627.749]: master offset          7 s2 freq  +40527 path delay      6535
ptp4l[2628.749]: master offset         56 s2 freq  +40578 path delay      6545
ptp4l[2629.749]: master offset         61 s2 freq  +40600 path delay      6545
ptp4l[2630.750]: master offset         67 s2 freq  +40624 path delay      6550
ptp4l[2631.750]: master offset        -77 s2 freq  +40500 path delay      6543
ptp4l[2632.750]: master offset        172 s2 freq  +40726 path delay      6547
ptp4l[2633.750]: master offset         17 s2 freq  +40623 path delay      6542
ptp4l[2634.750]: master offset        -88 s2 freq  +40523 path delay      6538
ptp4l[2635.750]: master offset        -55 s2 freq  +40529 path delay      6538
ptp4l[2636.750]: master offset        157 s2 freq  +40725 path delay      6538
ptp4l[2637.750]: master offset         71 s2 freq  +40686 path delay      6536
```

Figure 4-5: Log information obtained from the PTP daemon in the WIFI-TSN AP

In the “master offset” column the synchronization error (in ns) that the PTP daemon has measured every second can be observed. Similar to the error measured with the PPS signals it is bounded to a +-180 nm margin. In this case, the mean is around 0 so the PTP daemon seems not to be aware of the aforementioned 2.7 us error.

Additionally, we wanted to compare the performance impact by using a Z16 initially, and then using a modified system with enhanced oscillators to reduce jitter: the Z16 Low Jitter (Z16-LJ).

In the histogram of Figure 4-6, we can see that the differences in PPS between TSN WiFi Access Point and Low Jitter are localized in a smaller range and follow a normal distribution, while the differences in PPS between the regular Z16 and TSN WiFi AP exhibit more dispersion and a more erratic distribution.

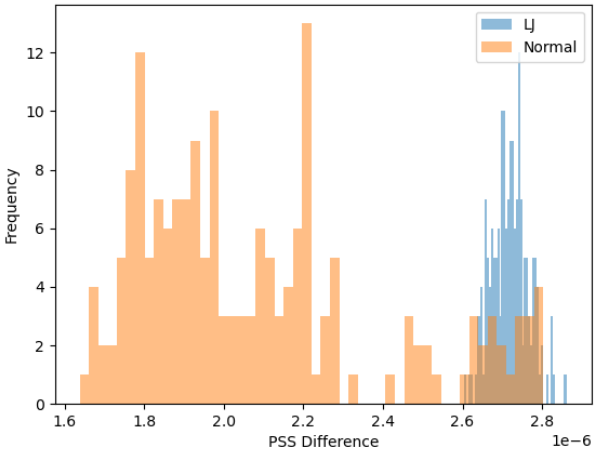


Figure 4-6: PPS Differences histogram

In Figure 4-7, we compare the calculated values of mean, maximum and minimum differences, range of values, and standard deviation between results using regular Z16 and Z16 Low Jitter. We can observe that the values of PPS differences are higher using Z16 Low Jitter than using regular Z16 (both the mean differences and maximum and minimum values). This aligns with the information obtained from the histogram, where we can see that the values using Z16 Low Jitter are shifted further to the right on the x-axis. However, by examining the range and standard deviation, we can see that the dispersion of differences using Z16 Low Jitter is much lower, just as in the histogram where the data is situated in a smaller range.

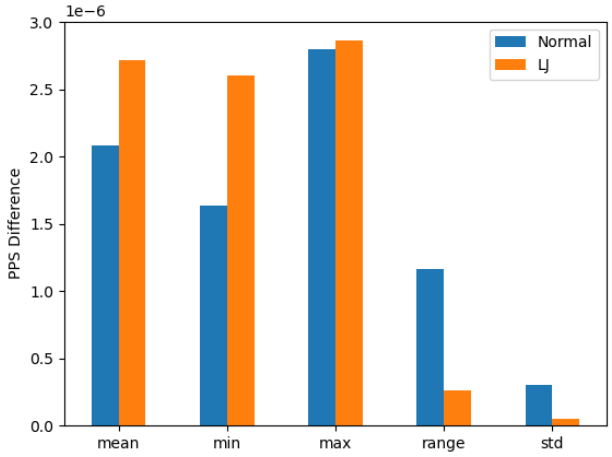


Figure 4-7: PPS Differences

The obtained values can be seen in the table below.

Results		
	TSN WiFi AP - Normal Z16	**TSN WiFi AP - Z16 LJ**
Mean	2.0834e-06	2.71904e-06
Min	1.6378e-06	2.60444e-06
Max	2.8027e-06	2.86341e-06
Range	1.1649-06	2.58965e-07
Std	3.0625e-07	4.75586e-08

We can use the Allan deviation plot in Figure 4-8 to further quantify the differences between both devices – the regular Z16 and the enhanced Z16-LJ node – in terms of frequency distribution stability in the short term. Initially, both sets of measurements show increasing deviation with increasing τ, which is typical as longer time intervals can introduce more variation. However, the Z16 LJ graph shows a decrease in deviation at a certain point, indicating a period where the PPS differences are less variable or more stable. This implies that the PPS differences stabilize at a certain point for Z16 LJ, whereas we can see that the same does not happen with Z16.

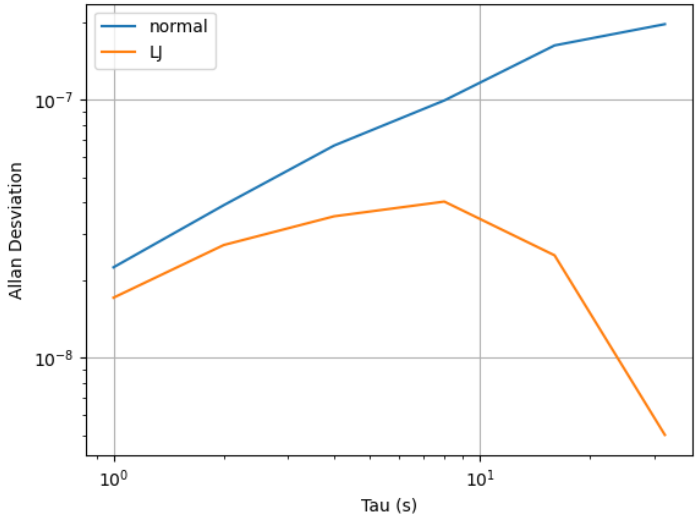


Figure 4-8: Allan Deviation

The main takeaway from these tests is that we have shown that the PTP stacks available on Safran's TSN switches and the IKERLAN TSN WiFi AP prototype are essentially compatible and capable of propagating a unified time reference between the respective network segments handled by Safran and IKERLAN in the demonstrator scenario of TIMING. These PTP tests show an early, preliminary integration that showcases how this time reference can be distributed between the Safran and IKERLAN network domains with different levels of performance as a function of the available hardware when the default profile of PTP over UDP with unicast connections and end-to-end (E2E) transport is used. Other profiles could potentially work as well and be considered for this integration. Nonetheless, this aspect will be further investigated in future integration tests in the upcoming stages of the project.

5 CONCLUSIONS

This deliverable reports the progress on the implementation of the different software components/modules that compose the TIMING architecture. The validation of the different modules in a stand-alone scenario is the prior step of the proper integration; some preliminary integration tests also reported. The results reported show a significant and solid progress in the implementation work that well position the project towards the final integration and validation.

Moreover, an operational workflow for E2E service provisioning has been also defined and reported. From it, the different steps to materialize the provisioning of a service with specific KPIs requirements over the E2E TIMING infrastructure have been identified as well as the different interactions among the software components of the architecture. This is a significant step achieved towards pushing the overall integration, prior to the architecture validation to be performed in the framework of SP3.

6 REFERENCES

[1] TIMING Project, Deliverable SP1 D1.2 “Year 1 Report on Requirements, Architecture and Performance Evaluation”, 2023.

[2] TIMING Project, Deliverable SP1 D1.3 “Preliminary version of software components”, 2023

End of Document